

NewBrain

BEGINNERS GUIDE

NewBrain

BEGINNERS GUIDE

SECTIONS

1.	Getting started	5
2.	Loading a program from tape and running it	8
3.	Starting programming	18
4.	Interaction	26
5.	Loops	34
6.	Organisation and presentation	39
7.	Saving programs on tape. Inspection of programs	48
8.	Graphics characters and screen	54
9.	Editing	62
10.	Strings	64
11.	String handling	69
12.	String handling to solve a problem	73
13.	Arrays (1)	81
14.	Arrays (2)	88
15.	Graphics (1)	95
16.	Graphics (2)	101
17.	Data	108
18.	RND, INT, etc.	110
19.	Data files on tape	114
20.	Extras and expansions	120
ERROR MESSAGES		124
INDEX TO INFORMATION PANELS		130

SECTION 1—GETTING STARTED

The aim of this Guide is to enable you to learn in a practical way by doing things. So we are starting with some straightforward, practical information and advice on how to set up your NewBrain system.

YOU NEED

- your NewBrain microcomputer, power supply and leads;
- a television set, or a video monitor;
- a cassette recorder, preferably one with a remote microphone control socket (usually labelled REM);
- cassette tapes: — the one supplied with this Guide;
— a blank cassette of good quality.

TELEVISION SET OR VIDEO MONITOR

You can use either a domestic television set or a video monitor with the NewBrain. You can even use both together, although this may cause some loss of picture quality. Leads are available for both, and the connections on the back of your NewBrain are labelled: — UHF for television set;
— MONITOR for video monitor.

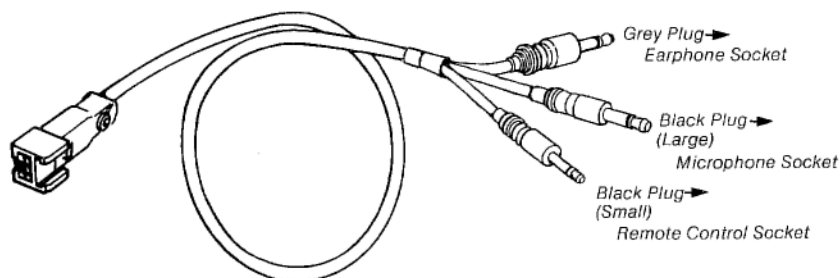
CASSETTE RECORDER

To use the Beginner's Guide effectively you need a cassette recorder. It is, of course, possible to use the NewBrain without one, but if you write programs of any length you will soon find that you want to save them for future use, and the cassette recorder provides an easy and cheap way of doing this.

You should, if at all possible, choose a good quality portable cassette recorder with a remote microphone control socket. (This is the small socket usually located beside the microphone input socket and usually labelled 'REM'.) You will

then be able to control the recorder from the computer keyboard. However, the remote control socket is not absolutely essential, and instructions are given in the Guide for recorders not so fitted.

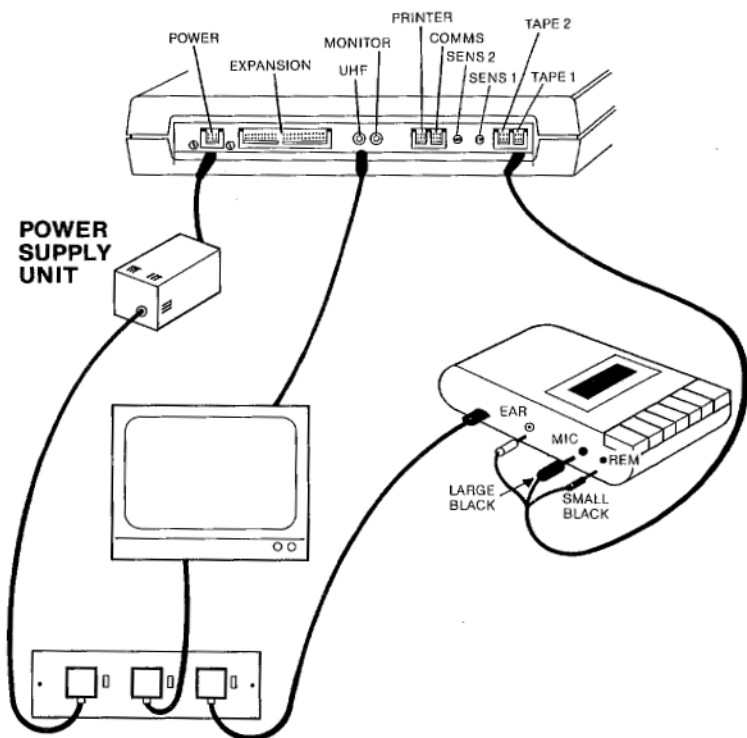
The Cassette Recorder Lead



CASSETTE TAPES

These should be ferric cassettes, C60 or shorter, of high quality manufacture. It is well worth paying a little more for your cassettes to avoid unnecessary frustration.

PANEL 1-CONNECTING UP



First connect everything up, but do not switch on yet. When switching on, either

1. switch everything on at once, i.e. at the mains, with the television set already switched to ON.

or

2. make sure the computer is switched on last. The reason for this is that any surge of current caused by switching may affect the computer.

Having switched on, wait for about 10 seconds. (If you have the AD model, the built-in visual display will flicker, then blank.) Adjust the television set's tuning to about channel 36, and you should see this:

**NEWBRAIN BASIC
READY**

**IF YOU GET NO SCREEN MESSAGE
CHECK ALL CONNECTIONS
AND TRY AGAIN**

SECTION 2—LOADING A PROGRAM FROM TAPE AND RUNNING IT

Load, program, run — all are common enough words, but in computing each has a special and quite precise meaning.

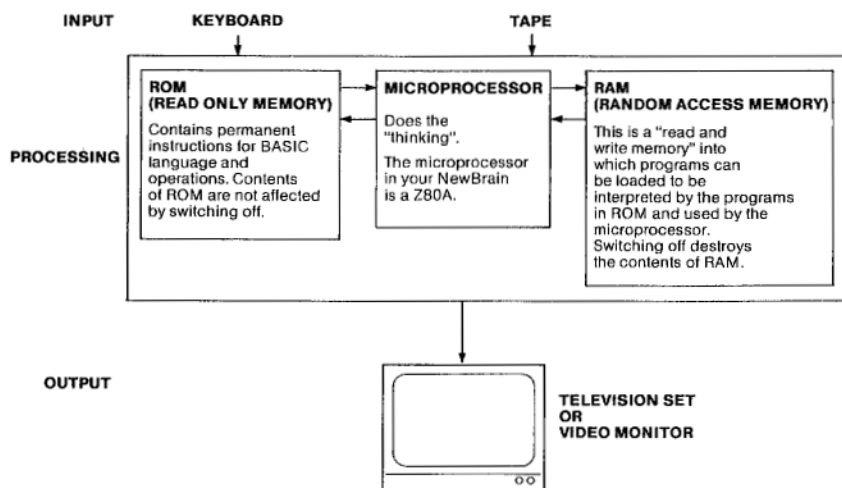
PROGRAM

This is a set of instructions for a computer to follow. In a sense it is a plan of action to solve a problem, but the plan incorporated in a program is completely inflexible. Once the program is in the computer and you tell the computer to implement it, the computer will do exactly what the program says, no matter how absurd the result may be. The computer is fast, precise and obedient, but it has no judgement — that's for you to provide!

Programs can be put into the computer in two ways: you can type them in on the keyboard; or you can load them from cassette tape. Once a program is held in the computer, you can change it as much as you like, using the keyboard.

LOAD

When we say that programs are “put into the computer” once again we mean something quite precise. To understand this you need to have a general idea of how your computer system is made up.



The easiest way of thinking of all this is in terms of input, processing and output. Input to the NewBrain is by keyboard or tape. Output is to a television set or a video monitor, or both together. (You can also output to a printer).

At the heart of the processing is the Z80A microprocessor, but this works at so primitive a level that most of us never even notice it. For most of us, the language of microcomputers is BASIC, which is a high-level language containing many words and commands that closely resemble normal English, although it has a grammar of its own.

The computer, therefore, must interpret between the low level microprocessor language and the high level BASIC.

The programs which do this interpretation are held in ROM devices, as shown in the diagram. In fact all the programs associated with the management of the computer's working are held in this way. ROM devices are not affected by switching the computer on and off, and the programs permanently stored in them save you, the user, a great deal of time and effort.

RAM devices are the opposite of ROM in one respect: if you switch off you lose whatever was stored in them at the time. What normally is stored in them is a program which you have put there for the computer to use. The NewBrain has 32K of RAM fitted as standard, and only a little of this has to be used in the computer's internal management, so lengthy programs can be entered and used. If later you find you need more RAM, more can be added, up to 2 megabytes.*

Loading from tape, therefore, means taking a program from tape and putting it into RAM.

RUN

This is a command in BASIC which tells the computer to start using the program.

ACTION

Enough of these explanations! It is time to LOAD a PROGRAM and RUN it.

1. Connect up the computer as in Panel 1.
2. Place the cassette in the tape recorder.

We are going to ask the computer to search the tape, find the first program in it, tell us what that program is called, load the program, and run it.

3. Set the volume control of the tape recorder to maximum. If your recorder is so loud it distorts, you may have to turn the volume down later, but it is best to start with it high. If you have a tone control, set it mid-way for the first attempt.

4. Type load and press **NEW LINE**

(If you typed loaf instead, or made a similar mistake, just press

*The size of RAM is measured by its storage capacity in terms of bytes. A byte is 8 bits, and a bit is the smallest unit recognised by computers. To give an idea of scale, a single letter or digit is equivalent to a byte. Memory sizes are expressed in terms of 'K' (for thousand) or 'M' (for mega or million) bytes.

NEW LINE anyway. The computer will probably reply with **ERROR 55** which means "keyword misspelt". Then try again.

5. Wind back the cassette, if it is not already wound back.
6. Press **PLAY** on the cassette recorder.
7. **THIS IS WHAT SHOULD HAPPEN NOW**
 - a) The cursor disappears (see below).
 - b) After a few seconds, the name of the first program appears, which in this case is "**NIM**".
 - c) The screen flickers as the tape runs. Sometimes the tape recorder stops and starts again, but do not do anything until the cursor reappears.
 - d) Finally there is a pause of a few seconds while nothing happens; then the cursor reappears and the program is ready to run.

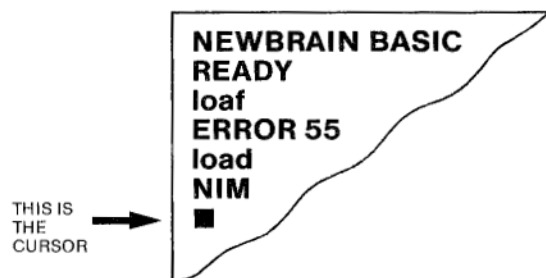
The whole process takes about 40 seconds.

IF YOUR CASSETTE PLAYER HAS A 'REM' CONTROL

— the tape stops automatically.

IF NOT — STOP THE TAPE YOURSELF by pressing **STOP** on the recorder.

After that your screen should look like this (assuming you have made one mistake).



IF ALL THIS HAS HAPPENED, GO ON TO 9

8. PROBLEMS

- a) Does the screen show **ERROR 130** or something similar? If you have a tone control, advance it to maximum and try again. Re-

check that all leads are properly connected and controls properly set. If you continue to have loading problems, suspect a fault — the NewBrain should load very easily. Try borrowing another cassette recorder to check on your own — quality is important here.

b) Does nothing seem to be happening at all?

STOP THE LOADING PROCESS BY PRESSING *

The program should load in about 40 seconds, even allowing for the leader tape at the start of the cassette. Check the volume control — is it too low? Use an ear-piece to check that you are getting some output from the ear-piece socket. Try varying the tone control between middle and maximum treble.

9. YOU ARE NOW READY TO RUN THE PROGRAM.

Press stop on the tape recorder to relieve pressure on the pinch roller.

Type `run` and press NEW LINE

The program is a version of the well-known mathematical game of NIM. Those who have met it before know that it is rather too predictable to be much of a game, but for us that is a virtue: we shall use it to understand programming. Notice the way in which the program is controlled through the keyboard.

Play it over a few times, to give you a feel for the NewBrain. If you should stop the program by mistake, there is no need to load it again: just type `run` and press NEW LINE.

NOTE: the NIM program uses black text on a white background, so you may find you have to adjust the contrast and brilliance on your television set.

PANEL 2—LOADING FROM TAPE

1. Connect up as in Panel 1: SWITCH ON
2. Place cassette in recorder
Set volume to maximum
Set tone to midway
3. Type load "program name" and press NEW LINE
4. Wind back the cassette, if it is not already wound back.
5. Press PLAY on the cassette recorder
6. Wait for the cursor to return. When it does, the program is.....
loaded.
IF YOUR RECORDER HAS A REMOTE CONTROL SOCKET
the tape stops automatically.
IF NOT — stop the tape yourself by pressing STOP on the
recorder.
7. Type run and press NEW LINE to start the program working.

LOADING ERRORS

If you get error messages and the program has not loaded, the most likely reason is that the volume control on your recorder is set wrongly. Experiment a little, but make sure you use a good cassette recorder.

NOTE: to stop the tape from the keyboard, press *

PANEL 2 – CONTINUED

.....ON MODEL AD THE BUILT-IN LINE DISPLAY FLICKERS

IF YOU DON'T SEE THIS

- your television set may not be adjusted properly;
- the leads may not be plugged in correctly.



**NEWBRAIN BASIC
READY**

..... *Recorder with REM control.* The tape may twitch a little when you press PLAY but it should not move forward until you type load. If it does, check leads and recorder; wind back and try again.

..... WHAT SHOULD HAPPEN ON LOADING

- a) Cursor disappears.
- b) Name of first program appears.
- c) Screen flickers as tape runs. Tape stops.
- d) When program is loaded, cursor re-appears.



**NEWBRAIN BASIC
READY
load
NIM**

TYPING ERRORS

If you type words such as load or run incorrectly, just press NEWLINE. You will get ERROR 55 or ERROR 26 on the screen, which means that the computer does not recognise your instructions. Then type the word correctly.

From now on we shall go into programming, proceeding by relatively easy stages.

At this point, many people have a secret worry that some mistake of theirs will do untold harm to their new computer. So we would like to give you the following assurance.

**NOTHING YOU TYPE ON THE KEYBOARD CAN EVER
DAMAGE YOUR NEWBRAIN MICROCOMPUTER.**

THE KEYBOARD

The NewBrain keyboard is laid out in much the same way as a typewriter keyboard, with a few extra keys for computer functions.

Just pressing keys A to Z normally gives you lower-case (small) letters; and pressing the **SHIFT** key with the letters gives upper case (capitals).

SHIFT LOCK — i.e. switching the keys to capitals only — is achieved by holding down **CONTROL** and pressing **1**. To switch back to normal, hold down **CONTROL** and press **0** (zero).



BACK SPACE — to correct typing errors on the same line:
EITHER 1. Press **←** and type over the same space correctly; or
2. Hold down **SHIFT** and press **←** to wipe out what you have just typed.

Having assembled the system, loaded a program from tape, and run it, we are now in a position to go into how programs work.

SECTION 3—STARTING PROGRAMMING

First run the NIM program until the computer asks you if you want another game. Type N for 'NO' and press **NEW LINE**. The computer replies Goodbye! and the program comes to an end.

When this happens, the computer is "out of the program", i.e. the program is no longer in control of what the computer does, so you can take over and tell it to do other things. To start with something simple, let's look at that part of the program which places the word "Goodbye!" on the screen.

To do this, type list 590 and press **NEW LINE**

(If you type it wrongly, just do it again.)

The screen shows you

```
590 PRINT "Goodbye!"
```

PRINT is a word in the BASIC language, and it means "put whatever comes after this instruction on to the screen". The figure 590 is a line number: every statement in a program has a line number so that the computer can find its way through in the right order. (In BASIC, every instruction to the computer is called a *statement*.)

You can try out the PRINT command for yourself without using program line numbers.

Type print "Hello" and press **NEW LINE**

The rule is that you have to enclose the item you want on the screen in inverted commas, but the inverted commas do not appear when the item is displayed. Try a few more of your own.

Now try this.

Type x = 25 and press **NEW LINE**

Type print x and press **NEW LINE**

This time there are no inverted commas, because we do not want the computer to print *literally* what follows the command. We want it to print the *value* of the VARIABLE 'x'. We have already told the computer what the value is, and as you can see from the screen the computer has remembered it.

Type `print x * 5` and press **NEW LINE**

The result is 125, because * means 'multiply'.

Try a few more, using +, -, and / which means 'divide'.

You can even define another variable 'y' as well and multiply the two together, or mangle it with x into a complex formula. Provided you first tell the computer what the value is, it can work out the answer. If you ask it to print the value of a variable you have not defined, it assumes that value is zero. (Note that zero on computer screens is always 'Ø', not 'O' which is a letter. So when you are typing, remember to type Ø if you mean the number, and 1 (one) if you mean number 1. The computer cannot imagine that when you type lower case letter "l" and capital "O" you really mean number 10.)

CLEARING THE SCREEN

By now you must have a lot of unwanted material all over the screen. To clear it all, hold down **SHIFT** and press **HOME**.

ALTERING A PROGRAM

We said earlier that it is possible not only to write a program through the keyboard, but to alter it as well. A simple piece of editing would be to change that "Goodbye!" at the end of the NIM program into something else.

Type `list 590` **NL** (From now on we shall use **NL** to mean **NEW LINE**)

The computer replies

`590 PRINT "Goodbye!"`

Note that although you have been typing "print", the computer lists it as "PRINT". In fact, the computer is equally happy with either, as it is with "Print" or even "pRiNt"

The simplest way to change a line in a program is to type another one with the same number. Suppose you type:

`590 print "Goodbye — sucker!"` **NL**

(If you make a mistake in typing it, you can just press **NEW LINE** and do it again.)

Type `list 590` **NL** to check that you have got it right. Then run

the program again and see your new version displayed on the screen at the end.

WRITING A PROGRAM

It is never too early to start writing programs. Many programs are long and complex because we want computers to do difficult jobs for us, but in principle a program can be very simple.

Suppose we write a program for the very simple task of doubling numbers: producing an endless series of numbers, each of which is double the one before.

First we have to get rid of the program that is in the RAM now.

Type new **[NL]** This clears the RAM.

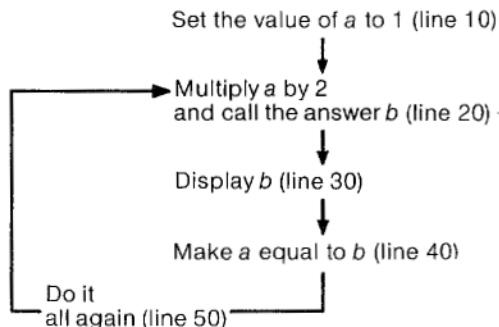
To check that the program is gone, type list **[NL]** or run **[NL]**. Now enter the new program. Type each line separately, pressing NEW LINE after each. (Yes, that's why it is called NEW LINE).

Again, if you make mistakes, just re-type the line. When you have finished, type list **[NL]** to check your work.

The program

```
10 a = 1
20 b = a *. 2
30 PRINT b
40 a = b
50 GOTO 20
```

What it means



The only new word in the programming is `goto` — which means simply “go to” and the number 20 means line 20. This is the main job of line numbers: to direct the computer to which command it should work on next. Incidentally, programmers usually start by writing their line numbers in tens because that leaves plenty of lines to spare for subsequent additions.

Now run the program. If it does not work and all you get is an error message or no result, list your program and check it very carefully to find your mistake.

What should happen is that the screen should be filled with numbers progressing upward, rapidly passing out of the computer’s range for ordinary numbers and appearing after that in “scientific notation”.

TO STOP A PROGRAM

Press the STOP key. The screen will tell you what line the computer was on when the program stopped.

A BIT MORE LOGIC

The program above does the job specified, but it is not the shortest and most economical that could be written to do that job. The BASIC language allows you not only to set the value of a variable, but to revise it as often as you like. So it is possible to write

```
10 a = 1
20 a = a * 2      (i.e. a = (old) a times 2)
30 PRINT a
```

so the new value of `a` is equal to the old value multiplied by 2

How would you finish the program after that? Have a try and see what happens when you run the result.

EXERCISE

Design a program to print on the screen a series of squares of numbers starting at 1. For example, 1 squared equals 1; 2 squared equals 4; 3 squared equals 9; 4 squared equals 16. So your series would begin 1 4 9 16 25 36 49 64 81 ... The answer is on the next page.

ANSWER

```
10 a = 1
20 b = a * a
30 PRINT b
40 a = a + 1
50 GOTO 20
```

SCREEN IMAGES

You probably noticed that when you switch on you get white characters on a dark background, yet the NIM program uses black on white. In fact you can select which set of characters you like from the keyboard.

1. Press **CONTROL** and hold it down while you press **W**
This gives you control over the character sets. Then ...
2. For black on white — press **C**
For white on black — press **B**

As you do this, you can take the opportunity to adjust your television set or monitor to display both white on black and black on white to best advantage.

PANEL 3—STARTING PROGRAMMING

To clear the screen

press **SHIFT** / **HOME** (i.e. hold down **SHIFT** and press **HOME**)

To clear a program from memory

new **NL** (**NL** = **NEW LINE**)

To start a program

run **NL**

To stop a program running

press **STOP**

To examine a program

list **NL** (lists the whole program)
list (line number) **NL** (lists that line)
list (line no. — line no.) **NL** (lists that block of lines)
e.g. list 10 — 100 **NL**

For black on white characters

CONTROL / **W** (i.e. hold down **CONTROL** and press **W**)
C

For white on black characters

CONTROL / **W**
B

Error messages: ERROR 55 = "keyword misspelt"
ERROR 26 = "keyword begins with something other than a letter"
(e.g. * print)

BASIC STATEMENTS

PRINT = display whatever follows this statement

- to display literally what follows, use inverted commas (**PRINT "ONE"**)
- to display the value of a variable, use the variable name (**PRINT a**)

GOTO = go to: continue program execution at the line specified.

PANEL 3 – CONTINUED

VARIABLES

Numeric variables work in much the same way as in algebra: the symbol stands for a quantity which may change according to variations in other quantities.

To set a value to a variable: just say what it equals

e.g. $a = z$ $a = b + 5$

A variable's value can be changed by adding to it. For example

$a = a + 5$ means (new) $a =$ (old) a plus five

TO CHANGE A PROGRAM LINE

Type the line again correctly, with the same number. The old line will be replaced automatically by the new one.

SECTION 4—INTERACTION

Getting the computer to display long series of numbers all by itself is good for showing its capability, but for best effect computers and people should work together.

The program below displays the fifth power of any number you enter.* It uses variables, as before, but this time instead of setting the value of the variable, the program asks the user to set it. Like this:

```
10 INPUT a
20 b = a^5
30 PRINT b
40 GOTO 10
```

Type the program into the computer. Notice the special computer sign (^) in line 20 for “to the power of” — you will find it sharing a key with the plus sign. Notice particularly the BASIC word INPUT. This causes the computer to place a question mark on the screen and to wait until you type a number and press NEW LINE. The number you type then becomes the value of a.

ERROR MESSAGES

When you try out this program, you will find that if you make a mistake and input a letter instead of a number, you get the message:

```
ERROR 30 at 10
?
```

ERROR 30 means simply an error on input, and the 10 is line 10. The question mark means that the computer is still at line 10 and is asking you to try again.

*If this is too much of a strain on your mathematical ability, the “fifth power” of 2 means $2 \times 2 \times 2 \times 2 \times 2$, which equals 32. But don’t let it worry you — this is the most complex mathematical idea in the whole *Beginner’s Guide*!

Run the program and try it out. If you try often enough you will soon notice that if you enter numbers above 39 the answer is given in scientific notation (e.g. If you input 40 the answer is 1.024E+08). This is because the NewBrain, like most computers, sets a limit on the range of numbers it will display in conventional notation. The NewBrain's usual limit is 99 999 999.

Suppose we decide that we will not accept answers in scientific notation: we will refuse all calculations that give an answer over 99 999 999. We now have to introduce a condition into the program — "if the answer is over 99 999 999 then do not display it".

One way of doing this is shown below:

```
10 INPUT a
20 b = a^5
25 IF b>99999999 THEN GOTO 50
30 PRINT b
40 GOTO 10
50 END
```

Study line 25 carefully: it contains one of the most useful and powerful constructions in BASIC. It means that if *b* is more than (>) 99 999 999 then the execution of the program should go on to line 50. BASIC allows you to write this line in several different ways, according to taste:

```
25 IF b > 99999999 GOTO 50
25 IF b > 99999999 THEN 50
```

All work equally well.

If you run the program as it stands, you will notice a disadvantage. With numbers up to 39 all goes well and the program returns to the beginning each time. With higher numbers, it just stops, and you have to type RUN to start it off again.

(To stop the program, hold down **STOP** and press **NEW LINE**)

EXERCISE

Change the program so that the computer displays a message to tell the user that his number is too large, whenever the result is over 99 999 999, and then returns to the beginning and asks for another input.

THE ANSWER IS ON THE NEXT PAGE

ANSWER

```
10 INPUT a
20 b = a^5
25 IF b>999999999 THEN GOTO 50
30 PRINT b
40 GOTO 10
50 PRINT "Your number is too large."
60 GOTO 10
```

Having got one piece of useful information onto the screen, how about some more, to make the program really “user friendly”?

EXERCISE

Alter the program to make it

- ask for a number;
- tell the user that he is being given the fifth power of that number.

Do not forget that you can add new lines in between those already there, and that you can replace lines by merely typing another line with the same number.

THE ANSWER IS ON THE NEXT PAGE

ANSWER — something like this:

```
5 PRINT "Type a number and press NEW LINE"
10 INPUT a
20 b = a^5
25 IF b>99999999 THEN GOTO 50
28 PRINT "The fifth power of the number is"
30 PRINT b
40 GOTO 5
50 PRINT "Your number is too large."
60 GOTO 5
```

For a smoother version of the program, load from the tape a program called "FIVE". Remember to type

load "FIVE" NL

NOT "Five" or "five". With names of programs this matters.*

Run the program and see how much clearer it is for the user. This is what is meant by "user friendly". The program is set out below.

```
10 PRINT: PRINT "Type a number"
20 PRINT: PRINT "and press NEW LINE"
30 INPUT a
40 PUT 31
50 b = a^5
60 PRINT "You entered";a
70 IF b > 99999999 THEN GOTO 100
80 PRINT "The fifth power of this is";b
90 GOTO 110
100 PRINT "That is too large for this program"
110 PRINT: PRINT: PRINT: GOTO 10
```

SEMI-COLON: IT MEANS "DO NOT GO
ONTO ANOTHER LINE"

PRINT alone
means a blank
line

THE COLON ALLOWS YOU TO PUT MORE THAN
ONE STATEMENT ON A LINE

THIS CLEARS
THE SCREEN

*Since the last program loops round itself, you may have difficulty getting out of it. There are two methods.

1. Switch off the computer or remove and replace the lead from the power unit. This destroys the program.
2. Hold down STOP and press NEW LINE. This gets you out of the program without destroying it.

Finally, in this section, let's turn to something lighter. By now you will have realised that your NewBrain has a full set of upper and lower case characters as well as many numbers and signs. Here is a program that lets you have a look at them.

```
10 x = 31
20 PRINT CHR$(x);
30 x = x + 1
40 IF x < 256 THEN 20
50 END
```

What is happening is that the program is asking for each character *by number*. Up to character number 127 the NewBrain character numbers follow the ASCII Code (American Standard Code for Information Interchange). Numbers 128 to 255 are special codes for graphics characters.

Type it in (carefully!) and run it. You may be surprised to notice that the program starts with $x = 31$, i.e. we do not make it ask for characters 0 to 30.

Can you guess the reason for this?

The best way to check on this is to print the characters' numbers beside them. We could also take the opportunity to improve the layout. Type

```
20 PRINT x; chr$(x); NL
```

Note the semi-colon.
This puts the character's
number (x) beside the
character.

Note the comma. This
makes a dramatic difference
to the layout, as you will
see when you run the program.

When you use a comma in a print statement you are in effect dividing the screen into four columns.

Try this for yourself.

PRINT "ONE" NL puts the word ONE at the left margin,
but PRINT , , "ONE" NL puts it half-way across.

With each comma you add you push the statement a set distance further across the screen. It helps if you imagine that the use of commas divides the screen into four invisible columns, each 10 characters wide. This can be very useful in setting out tables. (Incidentally, giving the computer separate commands from the keyboard like this, with no line numbers, does not affect the program in the computer in any way.)

If you run the program with the new line in it you will notice that the display begins with character 32 (which turns out to be a space). Where is character 31?

In fact `CHR$(31)` is not a character but another form of the "clear the screen" command we have used already in the form `PUT 31`. So when `x` starts at 31 it simply clears the screen, after which the other characters appear.

Characters up to and including 31 are what are termed *control characters*. They are means of controlling the screen rather than normal characters, and most of them cause no images to appear on the screen.

PANEL 4—MORE BASIC SCREEN LAYOUT

BASIC COMMANDS

- INPUT** Use it in the form: INPUT variable name (e.g. INPUT x)
 Places a question mark on the screen. Program execution stops until a number key is pressed followed by NEW LINE.
 You can stop the question mark appearing by using the form INPUT ("") x
- IF...THEN** — For example, IF x = 10 THEN 1010 (where 1010 is a line number)
 THEN may be replaced by GOTO or THEN GOTO.
 The line number may be replaced by an instruction, such as END or by a statement such as y = 3. (IF x = 10 THEN y = 3)
- :(colon)** May be used to divide several statements on the same line.

SCREEN LAYOUT

- PUT 31** clears the screen
- PRINT CHR\$(X)** where X is the ASCII or code number:
 places the character whose number is quoted onto the screen, unless the number refers to a control character.
- PRINT (by itself)** prints nothing — i.e. leaves a blank line
- ;(semi-colon)** inhibits NEW LINE. (NEW LINE is assumed after each print statement, i.e. the cursor moves down a line. This can be prevented by placing a semi-colon at the end of the statement.)

PANEL 4—CONTINUED

, (comma)

Moves the cursor to the next column to the right, or if the end of the line is reached onto the next line. This is a formatting aid: imagine the screen divided invisibly into 4 equal columns of 10 spaces each. Using a comma at the end of a statement moves the cursor (and hence the next item printed onto the screen) on to the next column.

CONTROL CHARACTERS

These are characters which control such things as clearing the screen (31), clearing a line (30), or moving the cursor up (11) or down (10). Most cannot be seen as actual characters on the screen.

ESCAPING FROM A PROGRAM WHILE INPUT IS OPERATING

Hold down **STOP** and press **NEW LINE**

SECTION 5—LOOPS

Think for a moment how you would design a program to input twelve numbers and work out their average, using the methods so far described in this Guide. (If you think you can actually program it — have a go!)

When you have thought about it, load the program called "Average" from the tape. (Type load "Average" **NL** — making sure you type the title exactly as it appears here.)

```
10 PUT 31
20 PRINT "This program works out the average"
30 PRINT "of twelve numbers." //////////////// ← Commas to provide 1½ lines spacing
40 x = 1: z = 0
50 PRINT "Enter number";x;"and press NEW LINE"
60 INPUT y
70 z = z + y: x = x + 1
80 PUT 31
90 IF x < 13 THEN GOTO 50
100 PRINT "Average of the 12 numbers is";z/12
110 END
```

Line 50 may be puzzling. Run the program and look at what happens. The semi-colon allows us to print the two pieces of text with the variable value in the middle, all on the same line. The point is that at the end of each PRINT statement the computer assumes a NEW LINE signal and moves onto a new line, unless the program tells it not to by inserting a semi-colon. (Numbers, incidentally, are always printed with a space or a minus sign before them, and a space after them.)

The program structure is a loop. Line 90 sends the computer round again until all 12 numbers have been entered. Note how the result is given in line 100.

In this case the loop is managed by the IF ... THEN statement, combined with GOTO, but there is another, usually more convenient way of designing loops. This is the FOR ... NEXT statement.

To see this in action, load "Average 2" from the tape. Run it and

you will see that its effect is identical with that of the original version.

```
10 PUT 31
20 PRINT "This program works out the average"
30 PRINT "of twelve numbers." ,,,,,,
40 z = 0
50 FOR x=1 TO 12
60   PRINT "Enter number";x;"and press NEW LINE"
70   INPUT y
80   z = z + y
90   PUT 31
100 NEXT x
110 PRINT "Average of the 12 numbers is";z/12
120 END
```

FOR ... NEXT loops work like this.

The FOR statement tells the computer how many times to go round the loop.

The NEXT statement tells the computer where the loop ends.

The instructions within the loop are carried out each time the loop is processed.

Once you get used to this method of constructing loops you will find it easy to recognise in a program, especially where there are loops within loops, and easy to construct too.

The variable x behaves just like any other variable, so we can print its value on line 60.

WARNING

```
10 FOR x = 1 to 10
20
30
40 FOR y = 10 to 15
50
60 NEXT y
70
80 NEXT x
```

This is correct. In fact you may "nest" any number of loops, up to the limit of the computer's memory.

```
10 FOR x = 1 to 10
20
30
40 FOR y = 10 to 15
54
60 NEXT x
70
80 NEXT y
```

This will not work. Crossing loops is bad logic and the computer will not accept it.

One common use of loops in a program is for timing. You have experienced one already in the NIM program, on lines 1200 to 1230.

```
1200 REM delay loop
1210 FOR z = 1 TO 600
1220 NEXT z
1230 RETURN
```

In NIM it was needed because otherwise everything would happen so quickly the game would become confusing.

Try altering the value (5000) in this program (or in the NIM program if you prefer) to see the effect different values have on the delay time.

```
10 PRINT "ONE"
20 FOR I = 1 to 5000
30 NEXT I
40 PRINT "TWO"
50 END
```

FOR ... NEXT loops may be varied at will. Instead of counting in ones, they can count in any interval you like including decimals. For example

```
20 FOR I = 1 TO 5000 STEP 10
```

will count in steps of 10. They can even count downwards, but if you do that you must not forget the STEP:

```
20 FOR I = 5000 TO 2000 STEP -5
```

The following program is called "Zig-zag". You will find it on the tape under that name, and it provides an interesting illustration of nested loops. The figures in brackets after the CHR\$ are codes for graphics characters. We shall deal with them later.

```

10  REM Zig-zag
20  PUT 23,67
30  FOR z=1 TO 10
40      FOR x=0 TO 11
50          FOR a=1 TO x+13
60              PRINT CHR$(146);
70          NEXT a
80          PRINT CHR$(255);
90          FOR a=x+15 TO 40
100             PRINT CHR$(146);
110          NEXT a
120      NEXT x
130  FOR x=10 TO 1 STEP -1
140      FOR a=1 TO x+13
150          PRINT CHR$(146);
160      NEXT a
170      PRINT CHR$(255);
180      FOR a=x+15 TO 40
190          PRINT CHR$(146);
200      NEXT a
210  NEXT x
220 NEXT z
230 END

```

Line 10 shows the programmer's friend: the REM statement. Anything with REM (= reminder) in front of it has no effect on the program, so you can write explanatory comments all over your programs if you wish to show how they work. This may seem totally unnecessary to you at this stage, but it is amazing how a program which sparkles with clarity and logic on the day you write it becomes as perverse and obscure as anybody else's program three weeks later.

You may be curious about line 20. This is a special code that sets up an alternative character set to provide the graphics characters used in the program (characters 146 and 255). We shall go into all that later.

PANEL 5-LOOPS

FOR ... NEXT

COUNTING UPWARD (an interval of 1 is assumed unless you state otherwise)

```
10 FOR I = 3 to 18  
20  
30
```

```
100 NEXT I
```

COUNTING UPWARD - STEPPED

```
10 FOR I = 1 to 100  
STEP 20  
20  
30
```

```
100 NEXT I
```

COUNTING DOWNWARD with minus STEP

```
10 FOR I = 300 to 1  
STEP -3  
20  
30
```

```
100 NEXT I
```

ALWAYS NEST LOOPS

NEVER CROSS THEM

```
10 FOR I = 1 to 10  
20 —  
30 —  
40 FOR X = 1 to 3  
50 —  
60 NEXT X  
70 —  
80 NEXT I
```

Any number of FOR ... NEXT loops may be nested, up to the limit of the computer's memory.

If the loop steps in integers (whole numbers, not decimals), the value of the variable (I above, for example) AFTER the loop is finished is always one step more than the limit indicated in the FOR statement.

For example, after FOR I = 1 to 10 NEXT I, the value will be 11

REM STATEMENTS

Anything typed on a program line after the word REM is disregarded by the program. Note, however, that REM statements occupy memory space, just like any other program line.

SECTION 6—ORGANISATION AND PRESENTATION

Did you notice anything strange about the Zig-zag program in the last section? Look at lines 50 — 110 and compare them with lines 140 — 200. Of course, they are the same. Before you conclude that programming must be a very boring business, you should know that there are ways of avoiding the boredom of programming things twice — to say nothing of the risk of getting it wrong the second time.

One of the most useful techniques in programming is designed to do this: it is the `GOSUB RETURN`. It works like this:

MAIN PROGRAM

10 —
20 —
20 `GOSUB 1000`
40 —
50 —
60 —
70 `GOSUB 1000`
80 —
90 —
100 —

SUB ROUTINE

1000 —
1010 —
1020 —
1030 —
1040 `RETURN`

`GOSUB` means "go to subroutine". `GOSUB 1000` means "go to the subroutine starting at line 1000". If you put any sequence that has to be repeated into a subroutine, you can use it as often as you like at any point in the program. It does not matter in the least where in the program you locate your subroutine.

In large programs, the main virtue of subroutines is to make the program easier to understand. If you set out the main building blocks of your program as routines to be called into action when you need them, it is much easier to think out what the program as a whole is doing.

At the end of a subroutine you place the word `RETURN`. `RETURN` has a quite precise meaning: `RETURN TO THE STATEMENT AFTER THE GOSUB WHICH SENT THE COMPUTER TO THIS SUBROUTINE`.

Note that word *statement*: if you have one statement per line in your program (as you would be well advised to do while learning to program), it will return to the next line. But if you have several statements on a line, separated by colons, it will return to the next statement on that line.

Like loops, subroutines can be “nested”; but it is just as important not to get your logic tangled up. If the computer encounters a RETURN without having gone through a GOSUB, all you will get is an error message.

EXERCISE

Change the Zig-zag program into a more professional form, by using a subroutine and cutting out the repetition.

THE ANSWER IS ON THE NEXT PAGE

ANSWER

```
10 REM Zig-zag 2
20 PUT 23,67
30 FOR z=1 TO 10
40   FOR x=0 TO 11
50     GOSUB 130
60   NEXT x
70   FOR x=10 TO 1 STEP -1
80     GOSUB 130
90   NEXT x
100 NEXT z
110 END
120
130 REM SUBROUTINE
140 FOR a=1 TO x+13
150   PRINT CHR$(146);
160 NEXT a
170 PRINT CHR$(255);
180 FOR a=x+15 TO 40
190   PRINT CHR$(146);
200 NEXT a
210 RETURN
```

GOSUB may be used quite freely. Use it in IF ... THEN statements if you wish:

IF X = 10 then GOSUB 1000

If X = 10 the program uses the subroutine before passing to the next statement. If not, it passes on to the next line.

COMPUTED GOTO

Another labour-saving device is the computed GOTO, but before we describe it you should see it in action.

Load the program on the tape called "Zodiac". The first display is an example of what computer people call a "menu" — i.e. a page from which you choose the item you want.

As you run it, try to imagine how its logic works. (NO PEEPING!) What kind of statements will be needed to run that menu?

You have probably concluded by now that the logic will be on the lines of:

```
100 INPUT X
110 IF X = 1 THEN 1000
120 IF X = 2 THEN 1500
130 IF X = 3 THEN 2000 .... and so on.
```

In fact, the steering is done in line 220. List lines 10—220 and you will see it all.

Line 160 asks for the number of the month. The computer remembers this, even though line 200 is asking for something else. Then line 220 uses the information.

```
10 REM Zodiac
20 OPEN#0,0: REM Switch to video alone
30 PUT 31: REM Clear the screen
40 PUT 23,87: REM Select black on white
50 PRINT: PRINT: PRINT" S I G N S    O F
█  T H E    Z O D I A C "/////////
60 PRINT TAB(6);"1.January"; TAB(22);
█  "7. July"
70 PRINT TAB(6);"2.February"; TAB(22);
█  "8. August"
80 PRINT TAB(6);"2.March"; TAB(22);
█  "9. September"
90 PRINT TAB(6);"4.April"; TAB(22);
█  "10.October"
100 PRINT TAB(6);"5.May"; TAB(22);
█  "11.November"
110 PRINT TAB(6);"6.June"; TAB(22);
█  "12.December"
120 PRINT: PRINT
130 PRINT"In which month were you born?"
140 PRINT
150 PRINT"Key number and press NEW LINE"
160 INPUT x
170 PUT 31: PRINT
180 PRINT"On what day of the month ";
█  "were you born?"/////////
190 PRINT"Key number and press NEW LINE"
200 INPUT y
210 PUT31:PRINT: PRINT      "          Y O U R
```

```

■      S I G N      I S"
220 ON X GOTO 230,260,290,320,350,380,41
■0,440,470,500,530,560

```

Line 220 means that if $X = 1$ the computer should go to the first line number listed; if $X = 2$ then it should go to the second; and so on. This method makes the use of menus very attractive to the programmer, as well as easy for the user.

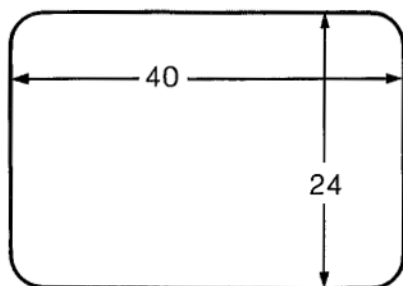
COMPUTED GOSUB

This works in exactly the same way as the computed GOTO. When the GOSUB has been executed, the computer returns to the statement after the computed GOSUB.

TAB

Whatever we do on the microcomputer, the result is usually apparent on the screen sooner or later. Many programmers concentrate on the logic and let the screen take care of itself, but this is a mistake, and may well be responsible for the feeling many people have that computers are obscure and hard to understand.

So far we have not paid much attention to the screen, apart from clearing it occasionally. But if you display a menu, you have to lay it out so that people recognise it for what it is. The TAB function is very useful here. The normal screen on your NewBrain is 40 characters wide by 24 deep.



The TAB command allows you to place words on the screen at any point along the line. For example:

```
PRINT TAB (6); "ONE" NL
```

places the word ONE on the next line starting at space 6. (Be careful not to forget the semi-colon.) In the Zodiac program there is a large collection of TAB commands to set out the menu.

One item at the end of the program may puzzle you. This is the S \$ on line 950. We shall have a good deal to say about that \$ in Section 10.

PUT 22

If TAB is useful in placing something at a precise point along a line, PUT 22 can do even more: it can place a character wherever you like on the screen. For example

```
PUT 22,10,15, "*" NL
```

places the * at the 10th space along the line and 15 lines down. Try it. (Note that you do not use PRINT in this case: PUT does it for you.)

You can also determine the point on the screen where the reply to an input is to be typed in, which may be useful if you are asking for information through your computer. Here is an example.

```
10 PUT 31: PRINT
20 PRINT "Type number here:"
30 PRINT: PRINT "and press NEW LINE"
40 PUT 22,19,2
50 INPUT ("")x
60 END
```


PANEL 6 - GOSUB, COMPUTED GOTO AND TAB

GOSUB ... RETURN

GOSUB sends the computer to a subroutine, which may be located anywhere in the program.

RETURN at the end of the subroutine sends the computer back to the next statement after the GOSUB.

COMPUTED GOTO

An example of this is

```
INPUT X  
ON X GOTO 100,200,300,400
```

It replaces a series of IF...THEN statements of the type:

```
IF X = 1 THEN 100  
IF X = 2 THEN 200 etc.
```

COMPUTED GOSUB

This works in a similar way to the computed GOTO and is even more useful, because if you use it well you can do without a lot of GOTOs, and program more efficiently.

```
100 ON X GOSUB 1000,1200,1300,1400,1500
```

TAB

TAB is used with print statements to place words at a precise point along a line. Thus

```
PRINT TAB(10); "ONE"
```

places the word ONE at the 10th space along the next line.

Caution: do not forget the semi-colon.

PANEL 6—CONTINUED

PUT 22

To place the cursor (and hence the next character to appear) at a chosen point on the screen, use

PUT 22,x,y

where x is a point along the text line (1—40) and y is the number of lines down from the top of the screen.

SECTION 7—SAVING PROGRAMS ON TAPE.

INSPECTION OF PROGRAMS

By now you have loaded several programs from tape. You are probably beginning to write your own, and soon you will feel the need to record them. To save your programs on tape, do the following.

1. Find a blank piece of tape (a blank cassette is best).
2. Make sure the cassette is wound past the leader tape.
3. (a) IF YOUR CASSETTE RECORDER HAS A REMOTE CONTROL SOCKET
 - set recorder to RECORD mode;
 - type save "program name" **[NL]** (For example: save "NIM")
- (b) IF YOUR RECORDER DOES NOT HAVE A REMOTE CONTROL SOCKET
 - type save "program name" BUT DO NOT PRESS NEW LINE*
 - set recorder to RECORD mode;
 - press NEW LINE immediately

The screen will flicker as the tape runs. When the cursor has returned the program should be saved.

(IF YOUR RECORDER DOES NOT HAVE REMOTE CONTROL: press STOP on the recorder immediately, or it will continue to run indefinitely.)

It is wise to check that your program has been saved correctly. Of course, you can simply LOAD it again, but if you do that you destroy your original program in RAM. So the NewBrain offers a VERIFY facility.

*If you are saving a long program — say a screenful or more — use the command save "**1: program name". This will save the program more slowly than normally, but should ensure success.

To verify a program

- type verify "program name" **NL**
- rewind the tape to a point before that at which the program starts;
- press PLAY on the cassette recorder.

The computer will compare the program on the tape with the program in its memory. If they are the same, it will display the message: VERIFIED. If not, it will print an error message.

VERIFY is also useful when you want to *fast forward* or *rewind* the tape on a recorder fitted with a REM socket. Most of these will not permit this while the REMote connector is plugged in, and you do not want to wear the socket by continually unplugging it.

If you type verify **NL**

the recorder is freed for *fast forward* and *rewind*. To cancel this condition, press *.

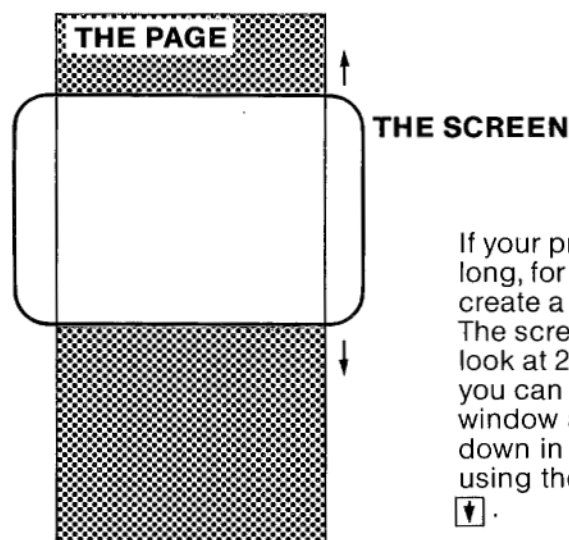
CATALOGUE OF PROGRAMS ON TAPE



To get a list of all the programs on your cassette, rewind it completely. Then ask the computer to load a program which does not exist, for example LOAD "O". The computer will search the whole tape, printing out the titles it finds on the screen.

INSPECTION OF PROGRAMS


It is easy enough to list a small program and inspect it as a whole, since it will all fit onto the screen. With larger programs this becomes tedious, since you have to ask the computer to show you a block of lines at a time.





The NewBrain overcomes this problem by making available not just a screenful of text at a time, but a whole "page", which can be very long indeed. It works like this.




If your program is 200 lines long, for example, you can create a "page" of 200 lines. The screen will only let you look at 25 lines at a time, but you can use the screen like a window and move it up and down in front of the page by using the editing keys  and .

To try it out, load a long program, such as NIM. (If you choose NIM, do not run it.) Then type (very carefully)*

open #0,0, "200" 

This gives you your "page" of 200 lines. Type LIST  and wait for the listing to end. Then press  and you are back at the start of the listing again, i.e. the screen "window" is looking at the first block of lines on the page. Hold down  to move downwards; and  to move back upwards again.

*If you type open #0 by mistake, your screen will go blank and pressing most of the keys will have no effect. To escape from this condition, press *

PANEL 7—SAVING AND INSPECTING PROGRAMS

TO SAVE PROGRAMS ON CASSETTES

1. Find a blank piece of tape. (A blank cassette is best.)
2. Make sure the cassette is wound past the leader tape.
3. (a) IF YOUR RECORDER HAS A REMOTE MICROPHONE CONTROL SOCKET
 - set recorder to RECORD mode;
 - type SAVE "program name" **[NL]**(b) IF YOUR RECORDER DOES NOT HAVE A REMOTE MICROPHONE CONTROL SOCKET
 - Type SAVE "program name" BUT DO NOT PRESS 'NEW LINE' YET
 - (IF YOU ARE SAVING A LONG PROGRAM use the command save "*1: program name" **[NL]**)
 - set recorder to RECORD mode;
 - press NEW LINE immediately.

The screen will flicker as the tape runs. When the cursor returns, the program should be saved.

(If your recorder does not have remote control: STOP it running immediately.)

TO VERIFY THE SAVING

- rewind the tape
- type verify "program name" **[NL]**
- set recorder to PLAY

TO FREE THE CASSETTE RECORDER FOR FAST FORWARD AND REWIND

Type verify **[NL]** To cancel this condition, press *

PANEL 7—CONTINUED

TO CATALOGUE ALL THE PROGRAMS ON THE TAPE

- type load "0" **NL** (i.e. ask it to find a program that does not exist)
- rewind the tape completely;
- press PLAY on the cassette recorder.

TO CREATE A "PAGE" AND INSPECT IT

- type open # 0,0,"200" **NL** (to create a 200-line page)
- list the program;
- to inspect the top block of lines, press **HOME**
- to move downwards, press **↓**
- to move upwards, press **↑**

SECTION 8—GRAPHICS CHARACTERS AND SCREEN

You must have realised by now that there is a good deal more to the NewBrain's handling of the screen than would appear at first glance.

When you switch on, you get large white characters on a dark background, yet when you run the NIM program, for example, you see black characters on white.








We mentioned earlier that you can switch from one to the other.

Black on white: press CONTROL/W followed by C

White on black: press CONTROL/W followed by B

Holding down Control and pressing W gives you control over several possible sets of characters on the screen.

Another possibility you may not yet have discovered is the set of Graphics Characters. Press the Graphics key, hold it down, and press any letter from A to Z. You should now be looking at a series of shapes. All these can be put onto the screen in programs and used to build up diagrams and pictures.

Try it out. A statement such as PRINT "        " will work just as well as with any other characters.

To give you an idea of some of the possibilities your NewBrain offers, load the program called "GR" from the tape and run it.

The interesting thing about the NewBrain's characters is that it is possible to exchange one set for another by changing the screen itself. The Panel at the end of this Section gives the control codes you need to do this, along with the codes to use in a program for the same purpose.

Screen switching, clearing lines, clearing the screen and moving the cursor can all be done by means of control characters. We have mentioned these before. The basic idea is of a set of characters which do not print, but cause certain actions to be performed by the computer instead. You are

using a control code whenever you hold down the Control key and press another key, or occasionally when you hold down two other keys.

For example, the control action for clearing the screen is SHIFT/HOME which is control character number 31. The command PUT is used to set a control character into operation, which is why we use PUT 31 to clear the screen in a program.

Look at line 20 of the "GR" program.

```
20 PUT 23,65,31
```

The 31 at the end clears the screen. Character 23 is in fact CONTROL/W which is what you press to gain control of the screen, and 65 is character A. So PUT 23,65 means CONTROL/W followed by A — and that means switch the screen to black on white.

One thing you may have found puzzling in the "GR" program is the small characters on 80-character lines. This is an alternative presentation which the NewBrain allows you to use in your program if you wish. To get it you type (or include in a program line)

```
OPEN # 0,0,"L"
```

This gives you a screen with 24 lines, each of 80 characters, and you can switch from white to black backgrounds as with the screen you are used to. "L" incidentally means "long", i.e. with a long line.

On some television sets, 80-column lines are difficult to read; television sets are not specifically designed for text.

To return to the normal screen with 24 lines of 40 characters, type

```
OPEN # 0,0 NL
```

```
or OPEN # 0,4 NL
```

You can add "S" (for short) to this if you wish, but there is no need to because "S" is what is known as the "default" value — i.e. the system assumes "S" unless you tell it otherwise. The difference between the two instructions above is that OPEN # 0,0 gives output to the television or monitor alone, and OPEN # 0,4 gives output to the display on the computer itself as well.

PANEL 8—CHARACTERS AND SCREENS

CHARACTERS	BACKGROUND	CHARACTER SETS	TO OBTAIN FROM KEYBOARD	TO USE IN PROGRAMS
WHITE BLACK	BLACK WHITE	1 1	CONTROL/W and D CONTROL/W and A	PUT 23,68 PUT 23,65
WHITE BLACK	BLACK WHITE	2 2	CONTROL/W and B CONTROL/W and C	PUT 23,66 PUT 23,67
WHITE BLACK	BLACK WHITE	3 3	CONTROL/W and H CONTROL/W and I	PUT 23,72 PUT 23,73
WHITE BLACK	BLACK WHITE	4 4	CONTROL/W and J CONTROL/W and K	PUT 23,74 PUT 23,75

It is also possible to print on the screen in reversed letters in blocks — as a way of emphasising the title of a program, for example. To try it out, select Character Set 1 (see table above) and type PUT 14: PRINT "REVERSE": PUT 15 NL

Note: if you should ever get stuck in this mode, press SHIFT/ESCAPE

MOVING THE CURSOR

PUT 11 moves the cursor up
PUT 10 moves it down

CLEARING PART OF THE SCREEN

The "GR" program clears an area at the bottom of the screen, as follows:

```
180 PUT 11, 11, 11, 11, 11, 11, 11: REM move cursor upward
190 PUT 1, 1, 1, 1, 1, 1, 1, 1, 1: REM insert blank lines
```

The cursor is moved up and then nine lines are inserted, all blank, so that the text below the cursor is pushed off the screen.

PANEL 8—CONTINUED

LONG AND SHORT LINES

OPEN # 0,0,"L" gives an 80-character line.

OPEN # 0,0 or OPEN # 0,4 gives a 40-character line.

TO CLEAR THE SCREEN: SHIFT/HOME or PUT 31

TO CLEAR A LINE: CONTROL/HOME or PUT 30

CHARACTER SET 1

0	1	2	3	4	5	6	7	128	129	130	131	132	133	134	135
8	9	10	11	12	13	14	15	136	137	138	139	140	141	142	143
16	17	18	19	20	21	22	23	144	145	146	147	148	149	150	151
24	25	26	27	28	29	30	31	152	153	154	155	156	157	158	159
	!	~	#	\$	%	&	/		!	~	#	\$	%	&	/
32	33	34	35	36	37	38	39	160	161	162	163	164	165	166	167
()	*	+	,	-	.	/	()	*	+	,	-	.	/
40	41	42	43	44	45	46	47	168	169	170	171	172	173	174	175
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
48	49	50	51	52	53	54	55	176	177	178	179	180	181	182	183
8	9	:	;	<	=	>	?	8	9	:	;	<	=	>	?
56	57	58	59	60	61	62	63	184	185	186	187	188	189	190	191
@	A	B	C	D	E	F	G	@	A	B	C	D	E	F	G
64	65	66	67	68	69	70	71	192	193	194	195	196	197	198	199
H	I	J	K	L	M	N	O	H	I	J	K	L	M	N	O
72	73	74	75	76	77	78	79	200	201	202	203	204	205	206	207
P	Q	R	S	T	U	V	W	P	Q	R	S	T	U	V	W
80	81	82	83	84	85	86	87	208	209	210	211	212	213	214	215
X	Y	Z	[\]	↑		X	Y	Z	[\]	↑	
88	89	90	91	92	93	94	95	216	217	218	219	220	221	222	223
\	a	b	c	d	e	f	g	\	a	b	c	d	e	f	g
96	97	98	99	100	101	102	103	224	225	226	227	228	229	230	231
h	i	j	k	l	m	n	o	h	i	j	k	l	m	n	o
104	105	106	107	108	109	110	111	232	233	234	235	236	237	238	239
p	q	r	s	t	u	v	w	p	q	r	s	t	u	v	w
112	113	114	115	116	117	118	119	240	241	242	243	244	245	246	247
x	y	z	{		}	~		x	y	z	{		}	~	
120	121	122	123	124	125	126	127	248	249	250	251	252	253	254	255

WHITE ON BLACK: PUT 23, 68: CONTROL/W and D

BLACK ON WHITE: PUT 23, 65: CONTROL/W and A

CHARACTER SET 2

0	1	2	3	4	5	6	7	128	129	130	131	132	133	134	135
8	9	10	11	12	13	14	15	136	137	138	139	140	141	142	143
16	17	18	19	20	21	22	23	144	145	146	147	148	149	150	151
24	25	26	27	28	29	30	31	152	153	154	155	156	157	158	159
32	33	34	35	36	37	38	39	160	161	162	163	164	165	166	167
40	41	42	43	44	45	46	47	168	169	170	171	172	173	174	175
48	49	50	51	52	53	54	55	176	177	178	179	180	181	182	183
56	57	58	59	60	61	62	63	184	185	186	187	188	189	190	191
64	65	66	67	68	69	70	71	192	193	194	195	196	197	198	199
72	73	74	75	76	77	78	79	200	201	202	203	204	205	206	207
80	81	82	83	84	85	86	87	208	209	210	211	212	213	214	215
88	89	90	91	92	93	94	95	216	217	218	219	220	221	222	223
96	97	98	99	100	101	102	103	224	225	226	227	228	229	230	231
104	105	106	107	108	109	110	111	232	233	234	235	236	237	238	239
112	113	114	115	116	117	118	119	240	241	242	243	244	245	246	247
120	121	122	123	124	125	126	127	248	249	250	251	252	253	254	255

WHITE ON BLACK: PUT 23, 66: CONTROL/W and B

BLACK ON WHITE: PUT 23, 67: CONTROL/W and C

CHARACTER SET 3

0	1	2	3	4	5	6	7	128	129	130	131	132	133	134	135
8	9	10	11	12	13	14	15	136	137	138	139	140	141	142	143
16	17	18	19	20	21	22	23	144	145	146	147	148	149	150	151
24	25	26	27	28	29	30	31	152	153	154	155	156	157	158	159
32	33	34	35	36	37	38	39	160	161	162	163	164	165	166	167
40	41	42	43	44	45	46	47	168	169	170	171	172	173	174	175
48	49	50	51	52	53	54	55	176	177	178	179	180	181	182	183
56	57	58	59	60	61	62	63	184	185	186	187	188	189	190	191
64	65	66	67	68	69	70	71	192	193	194	195	196	197	198	199
72	73	74	75	76	77	78	79	200	201	202	203	204	205	206	207
80	81	82	83	84	85	86	87	208	209	210	211	212	213	214	215
88	89	90	91	92	93	94	95	216	217	218	219	220	221	222	223
96	97	98	99	100	101	102	103	224	225	226	227	228	229	230	231
104	105	106	107	108	109	110	111	232	233	234	235	236	237	238	239
112	113	114	115	116	117	118	119	240	241	242	243	244	245	246	247
120	121	122	123	124	125	126	127	248	249	250	251	252	253	254	255

WHITE ON BLACK: PUT 23, 72: CONTROL/W and H

BLACK ON WHITE: PUT 23, 73: CONTROL/W and I

CHARACTER SET 4

0	1	2	3	4	5	6	7	128	129	130	131	132	133	134	135
8	9	10	11	12	13	14	15	136	137	138	139	140	141	142	143
16	17	18	19	20	21	22	23	144	145	146	147	148	149	150	151
24	25	26	27	28	29	30	31	152	153	154	155	156	157	158	159
32	33	34	35	36	37	38	39	160	161	162	163	164	165	166	167
40	41	42	43	44	45	46	47	168	169	170	171	172	173	174	175
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
48	49	50	51	52	53	54	55	176	177	178	179	180	181	182	183
8	9	:	;	<	=	>	?	8	9	:	;	<	=	>	?
56	57	58	59	60	61	62	63	184	185	186	187	188	189	190	191
Q	A	B	C	D	E	F	G	Q	A	B	C	D	E	F	G
64	65	66	67	68	69	70	71	192	193	194	195	196	197	198	199
H	I	J	K	L	M	N	O	H	I	J	K	L	M	N	O
72	73	74	75	76	77	78	79	200	201	202	203	204	205	206	207
P	Q	R	S	T	U	V	W	P	Q	R	S	T	U	V	W
80	81	82	83	84	85	86	87	208	209	210	211	212	213	214	215
X	Y	Z	[\]	^	_	X	Y	Z	[\]	^	_
88	89	90	91	92	93	94	95	216	217	218	219	220	221	222	223
l	L	+	r	7	J	+	h	a	b	c	d	e	f	g	
96	97	98	99	100	101	102	103	224	225	226	227	228	229	230	231
h	i	j	k	l	m	n	o	h	i	j	k	l	m	n	o
104	105	106	107	108	109	110	111	232	233	234	235	236	237	238	239
h	i	j	k	l	m	n	o	h	i	j	k	l	m	n	o
112	113	114	115	116	117	118	119	240	241	242	243	244	245	246	247
h	i	j	k	l	m	n	o	h	i	j	k	l	m	n	o
120	121	122	123	124	125	126	127	248	249	250	251	252	253	254	255

WHITE ON BLACK: PUT 23, 74: CONTROL/W and J

BLACK ON WHITE: PUT 23, 75: CONTROL/W and K

SECTION 9—EDITING

You already know that program lines can be changed by retyping them, i.e. typing a different line with the same number. You may also have become accustomed to the back-space facility. In fact, using all the editing keys



can save you a lot of time and trouble when you want to change items in a program. This is how you do it.

1. Move the cursor to the item you wish to alter, using the editing keys. As you move it, the cursor changes from ■ to —, which means that the computer is in EDITING mode. Positioning the cursor with the editing keys does not change any items on the screen.

2. TO REPLACE LETTERS

- type the new letters over the old ones;
- press **NEW LINE** and the line is amended.

3. TO INSERT LETTERS

- position the cursor so that its left-hand edge is at the point where the new letter is to be inserted;

10 PRNT "ONE"

- press **INSERT**;
- type in the new letters, and the old ones will space themselves out to receive the new material;
- press **NEW LINE** and the line is amended.

4. TO DELETE LETTERS





- position the cursor so that its left-hand edge is to the RIGHT of the letter you want to delete;

10 PRINOT "ONE"

- hold down **SHIFT** and press . The letter will vanish and the others will close up.
- press **NEW LINE** and the line is amended.

PANEL 9—EDITING

TO MOVE THE CURSOR

Press     Moving the cursor does not change any characters.


TO REPLACE LETTERS

- type the new letters over the old ones;
- press **NEW LINE** and the line is amended.

TO INSERT LETTERS

- position the cursor so that its left-hand edge is at the point where the new letter is to be inserted.
- press **INSERT**;
- type in the new letters, and the old ones will space themselves out to receive the new material;
- press **NEW LINE** and the line is amended.

TO DELETE LETTERS

- position the cursor so that its left-hand edge is to the RIGHT of the letter you want to delete;
- hold down **SHIFT** and press  The letter will vanish, and the rest of the letters will close up.
- press **NEW LINE** and the line is amended.

NOTE: some people find it easier to remember the general rule that you position the cursor to the right of the place where the insertion or deletion is to begin.

SECTION 10—STRINGS

In the “GR” program, you may have been puzzled by lines 230 and 510.

```
230 A$ = “WHITE ON BLACK — SAME GRAPHICS”
```

```
510 PRINT: PRINT A$
```

You have already become accustomed to *numeric variables* in expressions such as

```
10 INPUT A
20 IF A = 15 THEN GOTO 2000
```

Putting the dollar sign after the variable name shows that it is not a numeric variable, but a **STRING VARIABLE**, i.e. a variable which does not have a value in the sense that variable A above has, but which contains a *string of characters* — which may be letters, spaces, numbers, or graphics characters.

Try it out as you did with numeric variables.

```
A$ = “READ ONLY MEMORY” [NL]
Print A$ [NL]
```

and the computer obediently does so.

Note the inverted commas: they should always indicate to you that what is in a string variable is *literally* what is between those quotes.

You can ask for input to a string, just as with numeric variables:

```
10 INPUT A$
20 PRINT “THIS IS WHAT YOU ENTERED”; A$
30 GOTO 10
```

When you input, simply type in any letters or numbers. There is no need for quotation marks.

You can even add strings together, although computer people never talk of “adding strings”. They talk of “concatenation of

strings", i.e. putting them together, not adding, which might imply adding their values.

Run the program called "List" on the tape, to see the effect of concatenation.

```
10 REM List
20 FOR I = 1 TO 12
30   PUT 31: PRINT
40   PRINT "Enter name";I;"and press NEW LINE"
50   INPUT A$
60   B$ = B$ + " " + A$
70 NEXT I
80 PUT 31: PRINT
90 PRINT "Here is the complete list of names:"
100 PRINT: PRINT B$
110 END
```

Line 60 involves concatenation. It means "put together *what is already in B\$, with a space, and the new material in A\$, and call the result B\$*". The space is necessary to provide a break between the names, as the list is gradually built up.

Of course, the program is rather primitive in its presentation—that is no way to present a list! We shall have to give it some more attention later.

First we should deal with one more characteristic which string variables share with numeric variables: you can use them in IF ... THEN statements.

You can, for example, make your program ask for a word from the keyboard, and when it is typed in, check it against a list in the program itself. A password system could be constructed in this way.

Load "Zodiac 2" from the tape and run it. In this version, instead of presenting a menu the computer asks for the name of the month. It then checks what you have input against its own list.

```

10  REM Zodiac 2
20  OPEN#0,0: REM Switch to video alone
30  PUT 31: REM Clear the screen
40  PUT 23,87: REM Select black on white
50  PRINT: PRINT: PRINT "S I G N S    O F
   T H E    Z O D I A C "
60  PRINT "Type the name of the month"
70  PRINT:PRINT"in which you were born."
80  PRINT: INPUT X$
90  IF X$="January" OR X$="january" OR X
   $="JANUARY" THEN GOTO 230
100 IF X$="February" OR X$="february" OR
   $="FEBRUARY" THEN GOTO 260
110 IF X$="March" OR X$="march" OR X$="M
   ARCH" THEN GOTO 290
120 IF X$="April" OR X$="april" OR X$="A
   PRIL" THEN GOTO 320
130 IF X$="May" OR X$="may" OR X$="MAY"
   THEN GOTO 350
140 IF X$="June" OR X$="june" OR X$="JUN
   E" THEN GOTO 380
150 IF X$="July" OR X$="july" OR X$="JUL
   Y" THEN GOTO 410
160 IF X$="August" OR X$="august" OR X$=
   "AUGUST" THEN GOTO 440
170 IF X$="September" OR X$="september"
   OR X$="SEPTEMBER" THEN GOTO 470
180 IF X$="October" OR X$="october" OR X
   $="OCTOBER" THEN GOTO 500
190 IF X$="November" OR X$="november" OR
   $="NOVEMBER" THEN GOTO 530
200 IF X$="December" OR X$="december" OR
   $="DECEMBER" THEN GOTO 560
210 GOTO 80

```

Of course, there is a problem in comparing strings in this way. Computers can be very literal-minded, and if the programmer had used

```
IF X$ = "January" THEN 230
```

the computer would have ignored "JANUARY", and it is probably a good idea to look after the bad typist by including "january" as

well. That is what “user friendliness” is all about. One other point on user friendliness: suppose the user makes a complete mess of his typing and puts in something like “Jakllxxx”. Line 210 looks after this and causes the whole input program to start again.

Notice how useful the word OR is in this program. The word AND can be used similarly. For example

```
IF A$ = "JANUARY" AND T = 50 THEN 500
```


PANEL 10 - STRINGS

STRING VARIABLES

For example: A\$ X\$ A3\$ XP\$

The first item in the variable name must be a letter.

The second (if any) may be any letter or number, unless the result forms another BASIC word such as TO, ON, OR, IF.

The name must end in \$

STRINGS may be of any length, subject to memory available in the computer. They may be filled by

- a statement: e.g. A\$ = "TELEPHONE NUMBER IS"
- an INPUT: e.g. INPUT A \$
- concatenation: e.g. A\$ = B \$ + "AND" + C \$
or AS\$ = B\$ & C5\$

Strings may be used

- to print: e.g. PRINT X\$
- to compare: e.g. IF X\$ = "TELEX" then GOSUB 1000

NOTE: a command such as INPUT X\$ may be used in a program as a temporary stop, when for example you wish the user to read what is on the screen and press a key to go on. See line 950 of the Zodiac 2 program for such a use. The user presses NEW LINE to go on, inputting nothing.

AND / OR

These may be used in IF ... THEN statements, e.g.

IF X = 5 AND Y = 3 THEN 1000

IF X > 3 AND X < 15 THEN 1000 (i.e. if X is greater than 3 and less than 15)

IF X < 1 OR X > 5 THEN 1000

IF X\$ = "TELEX" AND Y >= 2 THEN 1000 (i.e. if Y is greater than or equal to 2)

SECTION 11—STRING HANDLING

Since the main output of your computer is onto the screen, it is necessary to have methods at your disposal to select the words you want, to place them where you want them and to order them appropriately. This involves string handling.

For example, we noted that the "List" program did not, in fact, produce much of a list. All the items came out in a long line with spaces between the words, but running over awkwardly from one line to the next. Curing that problem involves string handling.

BASIC has several methods. Try this for a start.

```
10 A$ = "ABCDEFGHJKLMNOPQRSTUVWXYZ"  
20 PRINT LEN(A$)  
30 END
```

LEN means length, and this gives the number of characters, including spaces, in the string. You may wonder why anyone should want to know that, but suspend judgement for a moment.

Three more BASIC functions are used in string handling:

MID \$
LEFT \$
RIGHT \$

You can try them out in the program above by replacing line 20 as follows:

```
20 PRINT MID $ (A$ ,5,3) — result: EFG
```

Start at the fifth character Print 3 characters



```
20 PRINT LEFT $ (A$,6) — result: ABCDEF
```

(Prints the 6 left-most characters)

```
20 PRINT RIGHT $ (A$ ,6) — result: UVWXYZ
```

(Prints the 6 right-most characters)

All these have been illustrated as PRINT statements, but they can equally well be included in such functions as

```
100 IF MID$(A$,5,6) = "LONDON" THEN 1000
```

or even

```
50 X=5
```

```
100 IF MID$(A$,X,6) = "LONDON" THEN 1000
```

Enough of the explanations! Load "List 2" and run it. Remember the items that you are entering for your list, because *there is a fault in this program*. The fault is that although you can input 12 items, the program only lists 11 of them.

THE EXERCISE IN THIS SECTION IS TO UNDERSTAND THE PROGRAM AND CORRECT THE FAULT.

First list the program and have a look. Try to understand the program for yourself at first. Note that up to line 80 it is much the same as the previous "LIST" program. The rest of the lines make up the vertical list.

NOT SURE WHAT IS WRONG? Then here is how the program works.

```
10  REM List 2
20  FOR I=1 TO 12
30      PUT 31: PRINT
40      PRINT "Enter name";I;
50      PRINT "(Only one-word names";
60      INPUT A$
70      B$ = B$ + " " + A$
80  NEXT I
90  PUT 31: y = 2
100 PRINT: PRINT "Here is the complete";
    " list of names",,,,,,
```

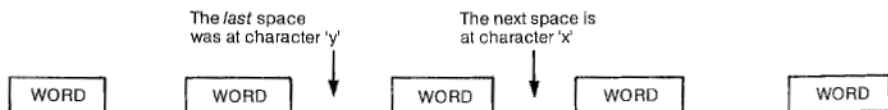
```

110 FOR x=y TO LEN(B$)
120 IF MID$(B$,x,1) = " " THEN GOTO160
130 NEXT x
140 END
150
160 PRINT"      "; MID$(B$,y,x-y)
170 y = x + 1: GOTO 110

```

Line 120 is looking for the spaces between the names. (Remember a space is a character.) When a space is found, the word *before* that space is printed at line 160

We need to examine the whole of B\$, but we do not know how many characters there are in it. So the loop ends at LEN(B\$)



So line 160 means

From character 'y'
PRINT 'X-Y' characters
PRINT MID\$(B\$,y,x-y)

After each item is printed, 'y' is reset to 'x' (the next space) in line 170, and the program goes back to find the next space.

NOW WHAT ABOUT THAT FAULT?

The point is that the list-printing routine depends on finding a space and printing the word before it. Item 12 ends the string, so there is no space after it and so it does not get printed.

The simplest remedy is to put a space after it.

85 B\$ = B\$ + " "

PANEL 11 – STRING HANDLING

- LEN(A\$) gives the length of A\$ in number of characters, including spaces.
- MID\$(A\$,X,Y) — gives Y characters within A\$ starting at character number X.
- LEFT\$(A\$,X) — gives the X left-most characters in A\$
- RIGHT\$(A\$,X) — gives the X right-most characters in A\$
- MID\$(A\$,X) — gives all the characters within A\$ starting at character number X.

These functions may be used in expressions such as

```
IF MID$(A$,X,2) = "PA" THEN 100
FOR X = 1 TO LEN(A$)
IF RIGHT$(A$,5) = C$ THEN 1000
PRINT MID$(A$,X,A-B)
PRINT MID$(A$,X,A-LEN(T$))
```

SECTION 12—STRING HANDLING TO SOLVE A PROBLEM

Suppose you are writing a program which is designed to test the user's knowledge. The program asks questions, and he has to type in the answers. The program then tells him whether he is right or wrong.

Suppose one of your question frames is like this.

In BASIC there are two types of variable.

One of them is numeric.

WHAT IS THE OTHER?

Type your answer and press NEW LINE.

?

By now you should have little difficulty in clearing the screen and setting up a layout like that. But what follows? An INPUT statement? Certainly — the word the user enters must be held in a string variable and checked against the correct answer which is held in the program itself.

Obviously the answer is "string", and we have already dealt with how to cope with "STRING" and "String". You simply compare each of them with the input variable, so that if the user answers with either of these we can tell him he is right.

But suppose he enters: "It is a string variable", "The other is a string variable", or merely "A String Variable". All would be quite reasonable answers, from a user's point of view, but a program worked out on the lines of

```
INPUT X$  
IF X$ = "string" then ....
```

cannot cope with them.

What we need is a routine which *searches the input string for the key word "string"*.

EXERCISE

Design a program to do just that. If the correct word is found, the screen should announce: "Your answer is correct". If not, it should tell the user: "Your answer is wrong".

THE ANSWER TO THE EXERCISE IS ON THE NEXT PAGE

If you find this too difficult, at least think about it, and program the easier bits at the beginning.

ANSWER

You will find this program on the tape. Its title is "Question".

```
10 REM Question
20 PUT 23,65
30 PUT 31: PRINT
40 PRINT "In BASIC there are two types"
50 PRINT "of variable.": PRINT
60 PRINT "One of these is numeric."
70 PRINT: PRINT
80 PRINT "WHAT IS THE OTHER?"
90 PRINT: PRINT
100 PRINT "Type the answer - ";
110 PRINT "then press NEW LINE": PRINT
120 INPUT X$
130 FOR I=1 TO LEN(X$)
140   IF MID$(X$,I,6)="String" THEN 210
150   IF MID$(X$,I,6)="string" THEN 210
160   IF MID$(X$,I,6)="STRING" THEN 210
170 NEXT I
180 PRINT: PRINT
190 PRINT "Your answer is WRONG": END
200
210 PRINT: PRINT
220 PRINT "Your answer is CORRECT": END
```

The routine at lines 130 to 170 steps along the string, character by character, matching each block of six characters against the word "string".

The routine shown above is in normal BASIC, but NewBrain BASIC goes one better. It has a special function which *detects the presence of one string within another*. This function is INSTR. Try the following.

```
X$ = "It is a string variable" [NL]
PRINT INSTR(X$, "string") [NL]
```

If you typed it correctly, you got the answer 9, meaning that "string" begins on the ninth character of X\$. If "string" had not been in X\$, then the answer would have been 0.

EXERCISE

Change the program, using the new, labour-saving INSTR command. The answer is on the next page.

ANSWER

You can find this on the tape under the title: "Question 2".

```
10 REM Question 2
20 PUT 23,65
30 PUT 31: PRINT
40 PRINT "In BASIC there are two types"
50 PRINT "of variable.": PRINT
60 PRINT "One of these is numeric."
70 PRINT: PRINT
80 PRINT "WHAT IS THE OTHER?"
90 PRINT: PRINT
100 PRINT "Type the answer - ";
110 PRINT "then press NEW LINE": PRINT
120 INPUT X$
130
140 IF INSTR(X$,"String") > 0 THEN 210
150 IF INSTR(X$,"string") > 0 THEN 210
160 IF INSTR(X$,"STRING") > 0 THEN 210
170
180 PRINT: PRINT
190 PRINT "Your answer is WRONG": END
200
210 PRINT: PRINT
220 PRINT "Your answer is CORRECT": END
```

Now suppose you want to check still further on the way the user is answering. Suppose you want to know whether he is putting a lot of words into his answers, so that you can give him a hint to concentrate on the key words and save himself a lot of typing. To do this we need to count the number of words in his answer.

EXERCISE

Modify the program so that, if he answers correctly, we not only tell him he is right, but add the information:

"There are X words in the answer."

If you have no idea as to how to do this, look at the foot of the page for a hint.

THE ANSWER IS ON THE NEXT PAGE

Hint: try making the program count the spaces between words, rather than the words themselves.

ANSWER

You can find this on the tape under the title "Question 3".

```
10 REM Question 3
20 PUT 23,65
30 PUT 31: PRINT
40 PRINT "In BASIC there are two types"
50 PRINT "of variable.": PRINT
60 PRINT "One of these is numeric."
70 PRINT: PRINT
80 PRINT "WHAT IS THE OTHER?"
90 PRINT: PRINT
100 PRINT "Type the answer - ";
110 PRINT "then press NEW LINE": PRINT
120 INPUT X$
130
140 IF INSTR(X$,"String") > 0 THEN 210
150 IF INSTR(X$,"string") > 0 THEN 210
160 IF INSTR(X$,"STRING") > 0 THEN 210
170
180 PRINT: PRINT
190 PRINT "Your answer is WRONG": END
200
210 PRINT: PRINT
220 PRINT "Your answer is CORRECT"
230 y = 1
240 FOR I=1 TO LEN(X$)
250   IF MID$(X$,I,1) = " " THEN y = y+1
260 NEXT I
270 PRINT
280 PRINT "(There are";y;"words in the";
290 PRINT " answer.)"
300 END
```

The routine at 240 — 260 counts the spaces between the words, not the words. Each time it finds a space, it adds one to variable y. Since y starts at one, the number of spaces equals the number of words.

PANEL 12—INSTR

INSTR — detects the presence of a string within another; gives the position of one string within another.

EXAMPLE 10 INPUT B\$

(Input is "ABCDEFGHIJKLMNOPQRSTUVWXYZ")

20 PRINT INSTR (B\$, "LMN")

(prints 12 — since L is the twelfth character)

INSTR may also be used in IF ... THEN routines:

30 IF INSTR (B\$, "LMN") = 12 THEN 1000

SECTION 13—ARRAYS [1]

Everyone knows that computers are very good at sorting and ordering information.

In this section, we shall look at a typical case: what one might call a "League Table", although the same techniques could be used for examination results, or the results of surveys or experiments.

What happens is this: names and points (or scores) are fed into the computer, and the computer produces an ordered list, with the highest score at the top, and the rest in descending order below it.

Load "League Table" from the tape and try it out, but please do not list it just yet! League Table asks for input on five teams and their scores, and displays the results in order.

Having tried it, how do you imagine it works? From your knowledge of numeric and string variables, you may imagine that the names are held in a set of variables: A\$, B\$, C\$, etc. and the scores in a parallel set: A, B,C, etc. It certainly could be done that way, but even loading up these variables using input statements would involve a lot of program lines. There would have to be INPUTs for every separate variable, since you cannot go on from A\$ to B\$ to C\$ by putting them into a loop and adding something each time. And as for sorting out the highest and putting them all in order — the mind boggles at the sheer number of program lines to be written.

Of course, a way has been found to overcome the problem: *arrays*. NewBrain BASIC has one-dimensional and two-dimensional arrays. A one-dimensional array is arranged like this:

Numeric Arrays A(1) A(2) A(3) A(4) A(5)

String Arrays A\$(1) A\$(2) A\$(3) A\$(4) A\$(5)

You can think of them as a series of pigeon-holes into which you put data. The great advantage is that the figure in brackets (the "subscript") can be replaced by a *variable*. So if, as in the *League Table* program, you have to load five numbers into five pigeon-holes, you can do it like this:

```
10 FOR I = 1 TO 5
20 INPUT A(I)
30 NEXT I
```

The only catch about using arrays is that before you use them you must remember to declare to the computer the maximum subscript value you intend to use. This is known as *dimensioning* the arrays and its purpose is to reserve the appropriate amount of memory space for them. For the example above, therefore, we need the line:

```
5 DIM A(5)
```

which means that the array will have a maximum of five *elements* (the correct word for pigeon-hole).

In fact, NewBrain BASIC saves you to a certain extent even if you forget to dimension the arrays. It assumes that all undimensioned arrays have 10 elements. However, it is still better to do the dimensioning yourself: why use up more memory than you need?

Now have a look at the "League Table" listing. Lines 50 to 190 are the INPUT loop, loading the names into N\$(1) to N\$(5), and the scores into P(1) to P(5). The rest of the program sorts the teams into order by scores and displays the results on the screen.

```

10  REM League Table
20  DIM N$(5),P(5)
30  PUT 23,65
40  REM Ask for teams and points
50  FOR I=1 TO 5
60      PUT 31: PRINT
70      PRINT "L E A G U E   T A B L E"
80      PRINT
90      PRINT"-----
■-----": PRINT
100     PRINT "Enter name of team";I
110     PRINT
120     PRINT "and press NEW LINE": PRINT
130     INPUT N$(I)
140     PUT 31: PRINT
150     PRINT "T E A M - ";N$(I)
160     PRINT:PRINT:PRINT"Enter score"
170     PRINT "and press NEW LINE": PRINT
180     INPUT P(I)
190 NEXT I
200
210 REM Present league table
220 PUT 31: PRINT
230 PRINT"L E A G U E   T A B L E"
240 PRINT
250 PRINT"-----
■-----": PRINT
260 PRINT "   TEAM";TAB(30);"POINTS"
270 PRINT
280 FOR I = 1 TO 5
290     IF P(I) > P THEN 310
300     GOTO 320
310     P = P(I): X = I
320 NEXT I
330 IF N$(X) = "" THEN END
340 PRINT "   ";N$(X);TAB(30);P(X)
350 N$(X) = "": P(X) = 0
360 P = 0: GOTO 280

```

QUESTION

What is the function of line 290?

ANSWER

In lines 290 and 310, every time a score ($P(I)$) is more than P , the variable P is set to that score, and the program remembers which score it was by setting X to the value of I .

When the loop is finished, therefore, $P(X)$ is the highest score, and N(X)$ is the team with that score.

Line 340 then puts the team and its score onto the screen.

Line 350 wipes the array element $P(X)$ and the team name N(X)$ and the program returns to the loop to search for the highest score among those left.

When line 330 detects an empty name element all names must have been listed, and the task is ended.

EXERCISE

The present program destroys the names and scores as it puts them onto the screen. How could you change the program to preserve them?

How would you change the program to add more points to scores already plotted on the screen: i.e. make the program go round in a circle, continually up-dating the results?

Program it if you can, but at least work out your ideas on how it could be done.

ANSWER

```

10 REM League Table 2
20 DIM N$(5),P(5),X$(5),Y(5)
30 PUT 23,65
40 REM Ask for teams and points
50 FOR I=1 TO 5
60     PUT 31: PRINT
70     PRINT "L E A G U E   T A B L E"
80     PRINT
90     PRINT"-----
■-----": PRINT
100    PRINT "Enter name of team";I
110    PRINT
120    PRINT "and press NEW LINE": PRINT
130    INPUT N$(I)
140    PUT 31: PRINT
150    PRINT "T E A M - ";N$(I)
160    PRINT:PRINT:PRINT"Enter score"
170    PRINT "and press NEW LINE": PRINT
180    INPUT P(I)
185    X$(I) = N$(I): Y(I) = P(I)
190 NEXT I
200
210 REM Present league table
220 PUT 31: PRINT
230 PRINT"L E A G U E   T A B L E"
240 PRINT
250 PRINT"-----
■-----": PRINT
260 PRINT "   TEAM";TAB(30);"POINTS"
270 PRINT
280 FOR I = 1 TO 5
290     IF P(I) > P THEN 310
300     GOTO 320
310     P = P(I): X = I
320 NEXT I

```

Load contents of N\$(I) into X\$(I), and contents of P(I) into Y(I) to preserve them

New arrays dimensioned

```

330 IF N$(X) = "" THEN 380
340 PRINT " ";N$(X);TAB(30);P(X)
350 N$(X) = "": P(X) = 0
360 P = 0: GOTO 280
370
380 REM Updating sequence
390 PRINT
400 PRINT "TO UPDATE RESULTS - PRESS ";
   "'NEW LINE'"
410 INPUT S$
420 FOR I=1 TO 5
430   PUT 31: PRINT: PRINT "TEAM:";X$(I)
440   PRINT:PRINT "PREVIOUS SCORE:";Y(I)
450   PRINT
460   PRINT "Enter additional points"
470   PRINT "and press NEW LINE": PRINT
480   INPUT P(I)
490   P(I) = P(I) + Y(I): N$(I) = X$(I)
500   Y(I) = P(I)
510 NEXT I
520 GOTO 210

```

Go back to display revised results

Ask for new input and
add to existing scores

The task of sorting information is one of the most common given to computers. It is also one of the most difficult, and many ingenious methods have been developed. The method used in *League Table* is the simplest and slowest: to go through the entire list each time. Faster methods, with names such as "bubble sort", "shell sort" or "quick sort" are used when there are thousands of items to be sorted.

PANEL 13—ARRAYS [1]

ONE-DIMENSIONAL ARRAYS work in the same way as numeric and string variables, except that they have subscripts which can be represented by variables.

EXAMPLE — arrays of 5 elements each

A(1) A(2) A(3) A(4) A(5)

A\$(1) A\$(2) A\$(3) A\$(4) A\$(5)

Arrays may have up to 5374 elements.

Each element in a string array may vary in length according to the program's requirements.

Subscripts may be varied numerically — for example

```
10 FOR I = 1 to 5
20 PRINT A$(I)
30 NEXT I
```

DIMENSIONING — Before any element of an array is used, the array must be dimensioned. If any element is used before dimensioning, the computer assumes the array's dimensions to be 10 elements.

TO DIMENSION — use the command DIM

```
10 DIM A (25)
10 DIM S$(20),A(6), C(101), D$(4)
```

ONCE AN ARRAY IS DIMENSIONED it must not be re-dimensioned, unless the dimensioning is cancelled by the command CLEAR. Note that CLEAR also removes the contents of the array, and of every other variable in use. CLEAR may also be used as a restricted command.

For example, to clear the contents of the array

R\$(1) R\$(2) R\$(3) R\$(4)

use CLEAR R\$ ()

SECTION 14 ARRAYS [2]

In the last section, we mentioned the computer's value in sorting and ordering information. The "League Table" is a simple example of this. By the use of arrays, it is a practical proposition to design systems that will accept information and display it according to different criteria. For example, instead of teams and points, we might design a program to collect information on products: price, quantity sold, income from sales, profit, cost — and the results could be displayed and ordered according to any of these criteria. That is what makes computers so useful in business.

But a good deal of the information that goes through computers is not in the form of a single row or column. Computers often have to process pages of accounts and statistics, in which lines and columns cross each other, and individual numbers relate to line and column headings in both directions in the form of a table. To cope with such arrangements, the NewBrain has TWO-DIMENSIONAL ARRAYS.

Our example in this section is taken from *cryptography*, or the making of codes. Most people tend to think of codes in terms of lists of letters accompanied by other symbols which are substituted for them in the coded message — A=P; B=S; C=E; and so on. In practice, many codes rely not on *substitution* but on *transposition*, i.e. the letters of the original message remain the same, but the order is changed according to a set plan, which can be reversed when decoding. In this section we are going to use two-dimensional arrays in programs which ENCODE and DECODE messages.

When the transposition method is carried out by hand it involves a good deal of writing. Suppose the message is: "MEET ME UNDER THE CLOCK AT WATERLOO STATION". To encode, you write it out in a table, missing out spaces:

```

M E E T M
E U N D E
R T H E C
L O C K A
T W A T E
R L O O S
T A T I O
N B V E T

```

Any spaces left in the matrix are filled in with nonsense letters.

The code is derived by reading the message in *columns* instead of in rows:

```
MERLTRTNEUTOWLABEDHCAOTVTDEKTOIEMECAESOT
```

The receiver of the message only needs to know the dimensions of the table to decode it. Obviously this is a gift for two-dimensional arrays.

Two-dimensional arrays are, in fact, very like the table above.

```

A(1,1) A(1,2) A(1,3) A(1,4)
A(2,1) A(2,2) A(2,3) A(2,4)
A(3,1) A(3,2) A(3,3) A(3,4)
A(4,1) A(4,2) A(4,3) A(4,4)

```

To dimension this array:

```
10 DIM A(4,4)
```

The array elements can be referred to using variables, just as with one-dimensional arrays.

Load the program "Encode" from the tape, and try it out. If you type in the message: "MEET ME UNDER THE CLOCK AT WATERLOO STATION ON THURSDAY AT TWO" you will get the message:

```

MDOEIRW*EECROSOUERKLND*NT**O*AMD*TA00YEE
MHT*N*EREE*S*AT***WTTT*TUCAAH*MHNLTUTTEE

```

Note that rather than miss out the spaces the program has replaced them with "*". On decoding these can be replaced by spaces again automatically.

Another subtlety of the program is that when it gets to the end of the message you have typed in, it fills up its matrix by *repeating as much of the message as will go in*. All coded messages are therefore of the same length: 80 characters or two lines of text. Since 80 characters are used, we can construct a table 8 by 10, and the dimensioning of the string array is

```
DIM A$(8,10).
```

```
10  REM Encode
20  DIM A$(8,10): X = 1
30  PUT 23,65
40  PUT 31: PRINT
50  PRINT"TRANSPPOSITION CODE"
60  PRINT: PRINT "ENCODING PROGRAM"
70  PRINT: PRINT
80  PRINT "Type in your message."
90  PRINT"Do not type more than 2 lines"
100 PRINT "Do not use any commas."
110 PRINT
120 PRINT"Ignore line endings. Type on"
130 PRINT"until you have finished the"
140 PRINT"message.": PRINT
150 PRINT "Then press NEW LINE": PRINT
160 INPUT A$: A$ = A$ + " "
170 PUT 31:PRINT
180 PRINT "THE ENCODED MESSAGE IS"
190 PRINT: PRINT
200 FOR I = 1 TO 8
210   FOR J = 1 TO 10
220     IF MID$(A$,X,1)=" " THEN A$(I,J)
230     = "*": GOTO 240
240     A$(I,J) = MID$(A$,X,1)
250     X = X + 1
260     IF X > LEN(A$) THEN X = 1
270   NEXT J
280 NEXT I
```

```

290     FOR I = 1 TO 8
300         PRINT A$(I,J);
310     NEXT I
320 NEXT J
330 END

```

Lines 200 to 270 work like this:

- X has been set to 1 in line 20
- On lines 210 — 260 the value of X is used to step along the string A\$. (NOTE: this is a string and has nothing to do with the array A\$(8,10).) As the loop steps along A\$ each character is loaded into a separate array element.
- The I-loop (lines 200 and 270) changes the first digit of the array subscript, and the J-loop the second; so by the time both loops are complete the whole table has been filled.

Note line 160: a space is added to the end of the string, so that when repeat characters are added on to the end they are separated from the main message by that space.

EXERCISE

Design the decoding program. (You guessed it, didn't you?)

(If you find that too difficult, at least examine "Encode" and form an opinion as to how the decoding would be done.)

ANSWER

The decoding program appears on the tape under the title: "Decode".

```
1000 REM Decode
1010 DIM A$(8,10)
1030 PUT 23,65: X = 1
1040 PUT 31: PRINT
1050 PRINT "TRANSPOSITION CODE"
1060 PRINT: PRINT "DECODING PROGRAM"
1070 PRINT: PRINT
1080 PRINT "Type in the encoded message."
1090 PRINT
1100 PRINT "Do not use NEW LINE until"
1110 PRINT "you get to the end of the"
1120 PRINT "message.": PRINT
1130 INPUT A$
1140 PUT 31: PRINT
1150 PRINT "THE DECODED MESSAGE IS:"
1160 PRINT: PRINT
1170 FOR J=1 TO 10
1180   FOR I=1 TO 8
1190     A$(I,J) = MID$(A$,X,1)
1200     X = X + 1
1210   NEXT I
1220 NEXT J
1230 FOR I = 1 TO 8
1240   FOR J = 1 TO 10
1250     IF A$(I,J)="*" THEN A$(I,J)=" "
1260     PRINT A$(I,J);
1270   NEXT J
1280 NEXT I
1290 END
```

"Decode" is, of course, virtually a mirror-image of "Encode". Note line 1250, which converts the "*" back into spaces.

EXPERIMENTS

Line numbers for Decode begin at 1000, so both Encode (which has line numbers from 10 to 330) and Decode could exist in the computer's memory at the same time. This would enable you to experiment with the coding system, encoding and decoding without the need to re-load programs continually.

But how do you load two programs at once? The command LOAD wipes out any existing program in memory. To avoid this problem, the NewBrain uses the command MERGE. It works like this.

1. Load Decode in the usual way.
2. Type merge "Encode"
3. To use Encode, type run
To use Decode, type goto 1030

(NOT GOTO 1000 since 1010 dimensions the array which has already been dimensioned in line 20. Arrays must not be re-dimensioned.)

You could even type goto 1230. This would feed the array A\$, which is already filled with the coded message, into the decoding sequence directly, without your having to type it again.

PANEL 14-ARRAYS [2]

TWO-DIMENSIONAL ARRAYS are in the form A\$ (X,Y), where X and Y identify the elements of the array. The arrays may be either numerical or string.

EXAMPLE A\$ (1,1) A\$ (1,2) A\$ (1,3) A\$ (1,4)
 A\$ (2,1) A\$ (2,2) A\$ (2,3) A\$ (2,4)
 A\$ (3,1) A\$ (3,2) A\$ (3,3) A\$ (3,4)
 A\$ (4,1) A\$ (4,2) A\$ (4,3) A\$ (4,4)

This is a 4 x 4 element array.

DIMENSIONING: this must be done before using any element.

USE: DIM (X,Y) where X and Y are the maximum values of the subscripts you wish to use. **EXAMPLE** DIM A\$ (4,4)

Where no dimension statements are included, the NewBrain assumes 10 x 10.

Arrays once dimensioned may not be re-dimensioned unless cleared.

Dimensioning may be cleared by using the command **CLEAR**. Note that **CLEAR**, cancels not only the dimensioning but also the contents of arrays. See Panel 13 for further information on **CLEAR**.

MERGE

This loads a program from tape without clearing the existing program from memory, and leaving the values of the variables intact.

Warning: if a line in the merged program has the same number as an existing line, the existing line will be replaced by the new line.

SECTION 15 GRAPHICS [1]

We had a look at graphics symbols a few sections back, but the NewBrain has much more to offer.

The high-resolution graphics facility allows you to display a special graphics screen on your television set or monitor, to "draw" on it in lines, curves or areas of black and white, and to place characters on it. You can even vary the size of the graphics "screen" to allow more or less room for normal text.

Load "Graphics Demo" from the tape. Run it and you will see everything promised in the last paragraph demonstrated, and more besides.

Designing good graphics is largely a matter of practice, once the basic methods are known. In this section and the next, therefore, we are concentrating on methods, which are summarised in the Panels. But do not forget that you can list the Graphics Demo itself and use it as a source for methods.

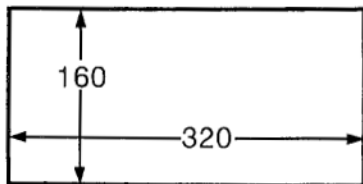
SETTING UP GRAPHICS

Type (very carefully!) open # 0,0, "110" NL

If you get it right, the small display on the computer will darken, and the screen will show white on black, whatever it was showing before. (Technically you are opening a "stream" down which data flows from the computer. In this case the numbers following "open" cause data to flow to the screen.)

Now type open # 1,11,"w160" NL

The "11" is graphics: "w" means a wide graphics screen which is 320 dots wide. "160" means 160 dots high. So our graphics screen is



but at the moment we cannot see it because it is in the same background colour as the rest of the screen and there is nothing plotted on it. Oddly enough we can detect its presence: try typing any letters at random on the screen and pressing NEW LINE and you will find you cannot get them to the bottom of the screen because they scroll after a few lines. That is because the graphics screen is there, although invisible.

Type plot background(1)
 plot wipe

You will soon get used to the word "plot" before all graphics commands. What you should now see is



(You can easily switch backgrounds, as with text, by using CONTROL/W and B or C.)

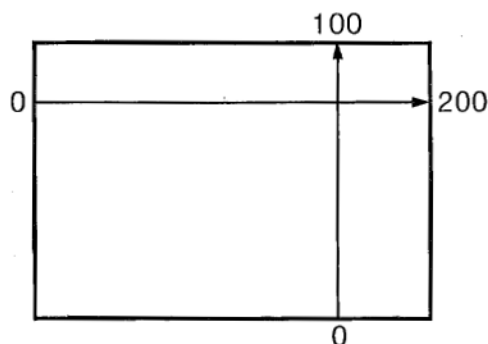
This is the graphics screen, but before we can draw on it we have to set its scale. NewBrain allows us to choose the scale. At the moment it is 320 by 160, a ratio of 2 to 1. So why not simplify matters by setting it to 200 by 100? This is done by typing

plot range (200,100)

We now have to plot a centre, which can be in the actual centre if we wish (i.e. 100, 50), or any other point we find convenient to start from. For this experiment, the choice is the bottom left-hand corner (0,0).

Type plot centre (0,0)

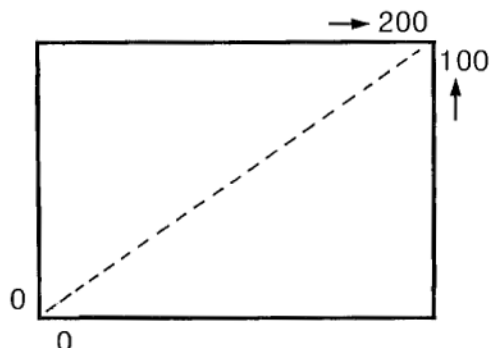
So we now have a screen with a scale of 0 — 200 from left to right and 0 — 100 from bottom to top.



Now to place the pen on the screen. There is, of course, no such thing as a pen, but it is convenient to imagine one. The command **PLACE**, for example, puts the pen on the screen, but does not cause it to make a mark. The command **MOVE**, on the other hand, moves it to another point, drawing as it goes.

Type plot place (0,0) **NL**
(i.e. put the pen in the bottom left-hand corner)

Type plot move (200,100) **NL**



Try a few more **MOVES** of your own. (Don't worry if you move the pen off the screen.) When you have finished, close down graphics and return to the complete text screen. To do this type

close # 1 **NL**

Now try the following program. Note how some commands have

been put together on the same line, using commas to divide them. This reduces the number of times you need to use the word PLOT.

```
10 OPEN#0,0,"100"  
20 OPEN#1,11,"n160"  
30 plot background(1),wipe  
40 plot range(200,100)  
50 plot centre(0,0)  
60 plot place(10,10)  
70 plot move(10,90), move(190,90)  
80 plot move(190,10), move(10,10)  
90 plot place(70,50)  
100 plot "FRAME"  
110 END
```

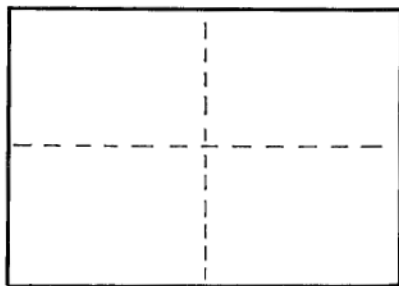
If you find it does not work, or if you get error messages, check the lines carefully. If you want to run it twice, you will find the computer objects to your trying to open #1,11 when the stream is already open. To avoid the problem, type close #1 **[NL]** before running. Alternatively you could write a new line:

```
5 close #1
```

Now try a graphics program for yourself.

EXERCISE

Write a program which places a cross on the screen like this (using as much of the present program as you like!).



THE ANSWER IS ON THE NEXT PAGE

ANSWER

```
5  CLOSE#1
10 OPEN#0,0,"100"
20 OPEN#1,11,"n160"
30 plot background(1),wipe
40 plot range(200,100)
50 plot centre(0,0)
60 plot place(0,50),move(200,50)
70 plot place(100,100),move(100,0)
80  END
```


PANEL 15—GRAPHICS [1]

GUIDELINES ON SETTING UP A GRAPHICS SCREEN

Height: for safety you should limit the height to 220 lines (screen dots)

You should choose heights in multiples of 10.

Width: the choice is 320 dots — “wide” (w)
256 dots — “narrow” (n)

EXAMPLE COMMANDS

10 open # 0,0, “150”

The “150” is the number of text lines on the page. Its function here is to reserve sufficient memory space for 220 lines of wide graphics.

20 open # 1,11, “w220”

This opens a wide graphics screen of 320 lines wide by 220 high

10 open # 0,0, “125”

This opens a narrow screen of 256 lines wide by 220 high

20 open # 1,11, “n220”

REMEMBER: the larger the graphics screen, the greater the demand it makes on the computer's memory, and the less room it leaves for your program.

TO CLOSE A GRAPHICS SCREEN — close # 1

TO CHANGE BACKGROUNDS — plot background(1)

(1 = opposite colour to rest of screen)

(2 = same colour as rest of screen)

TO CLEAR THE GRAPHICS SCREEN — plot wipe

TO PLOT LINES YOU SHOULD

1. SET THE SCALE — plot range (100,75)
(This is an example — you choose the range.)

2. SET THE CENTRE — plot centre (50,20)
(Again set whatever you find convenient: it does not have to be the true centre.)

TO PLACE THE “PEN” ON THE SCREEN WITHOUT MARKING
plot place (x,y)

TO MOVE THE PEN, MARKING AS IT GOES plot move (x,y)

Here x and y are the destination of the pen from where you last placed it or moved it.

SECTION 16 GRAPHICS [2]

The graphics facilities described in the last section enable you to plot lines wherever you like on whatever size of graphics screen you choose, and to vary the background colour too.

In this section, we shall go into NewBrain's special graphics facilities. First set up a graphics screen.

```
open # 0,0,"120" [NL]
open # 1,11,"w160" [NL]
plot background (1), wipe [NL]
plot range (200,100), centre (100,50) [NL]
plot place (0,0) [NL]
```

So now we have the pen in the centre, ready to move and draw lines. Last time we used the command MOVE, which moves the pen to a named location, drawing as it goes.

Now try this:

```
plot draw (50,50,1) [NL]
plot draw (50,-50,1) [NL]
plot draw (-40,-10,1) [NL]
```

Do you see what is happening? The pen is drawing lines to each location you name (50,50), but returning again after each line. The figure (1) is the "colour" — i.e. foreground colour. If you plot the background colour (2), the line disappears. To wipe out the first line we drew, type

```
plot draw (50,50,2) [NL]
```

Now although such facilities as these could be very useful — when drawing graphs for example — it would be even more useful to be able to "steer" the pen by giving it directions and distances.

All this is possible. Wipe the screen

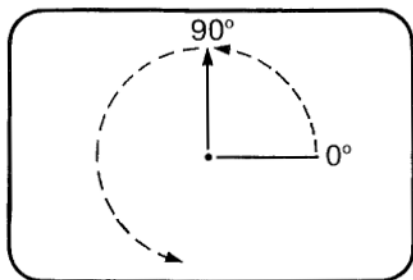
plot wipe [NL]

and try this

plot degrees [NL]

plot turn(90),drawby(40) [NL]

TURN moves the direction of the pen to 90° from its starting point at 3 o'clock.



Try a few more turns.

Then try this:

```
10 plot background(1),wipe
20 plot turn(0)
30 FOR I = 1 TO 36
40   plot turnby(10),drawby(40,1)
50 NEXT I
60 END
```

Turn sets the direction of the pen.

TURNBY (10) moves the direction through 10°

DRAWBY draws a line from the centre of length 40 in colour 1, and returns the pen to the centre.

This should produce a rotating pattern. If you want to change it so that it starts from the top rather than from the side, replace line 20 with

20 plot turn(90)

If you want the lines to go in a clockwise direction, use

40 plot turnby(-10), drawby(40,1)

Would you like to wipe out the previous line each time a new line is plotted, to give a rotating effect? Add the line

35 plot drawby(40,2) '2' is the background colour, so it wipes out the line

Would you like the pointer to make more than one revolution? Just take away the FOR ... NEXT loop and replace it with a GOTO. The following goes on for ever.

```
10 plot background(1),wipe
20 plot turn(90)
30 plot drawby(40,2)
40 plot turnby(-10),drawby(40,1)
50 GOTO 30
```

Now for some work. Load "Clock" from the tape. You will see it presents a format for two dials showing seconds and minutes.

EXERCISE — put the hands on the dials!
Or if that is too tough a proposition, put one of them on.

Hint: we found that the best place for the centres was (50,54) and (150,54) for the seconds and minutes respectively.

Here is a listing of "Clock". Before attempting the exercise, look at the program and at the notes that follow.

```
10 REM Clock
20 CLOSE#1: OPEN#0,0,"130"
30 OPEN#1,11,"w160"
40 plot range(200,100),centre(50,50)
50 plot degrees
60 plot background(1),wipe
70 PUT 23,65
80
90 REM Label the two dials
100 plot place(50,50),move(50,-50)
110 plot place(-20,-45),"Seconds"
120 plot place(82,-45),"Minutes"
130
140 REM Put numbers on seconds dial
150 plot place(-10,0),turn(90)
160 FOR D = 5 TO 60 STEP 5
170   plot turn(90-(D*6)),colour(2)
180   plot moveby(35),colour(1),D
190   plot place(-10,0)
200 NEXT D
```

```

210
220 REM Put numbers on minutes dial
230 plot centre(150,50)
240 plot place(-10,0),turn(90)
250 FOR D = 5 TO 60 STEP 5
260   plot turn(90-(D*6)),colour(2)
270   plot moveby(35),colour(1),D
280   plot place(-10,0)
290 NEXT D
300 END

```

TEXT ON GRAPHICS

You will have noticed the words "Seconds" and "Minutes" and the numbers printed on the graphics screen. The ease with which this can be done is a very useful feature of the NewBrain. The rules are

1. place the pen where you want the text to start;
2. use PLOT in the same way as you would use PRINT

For example:

```
plot place (50,50) NL
```

```
plot "Text" NL
```

In practice, the two commands can often be combined:

```
plot place (50,50), "Text" NL
```

Note also the command MOVEBY in line 180. This is a modification of MOVE, so the pen moves, drawing as it goes, but it moves only the distance indicated (35). This is a sequence for printing the numbers on the dials. What the program does is to move in background colour (so nothing is printed) from the centre to the number ring of the dials, then change the colour and print the numbers, which are the value at that time of the variable D, then move back to the centre again for the next number.

THE ANSWER TO THE EXERCISE IS ON THE NEXT PAGE

ANSWER

```
320 REM Set minute hand
330 M=96: plot centre(150,54)
340 plot place(0,0),turn(M)
350 plot drawby(25,2): M = M - 6
360 plot turn(M),drawby(25,1)
370
380 REM Set second hand
390 plot centre(50,54),place(0,0)
400 plot turn(90)
410 FOR S = 1 TO 60
420   plot drawby(25,2),turnby(-6)
430   plot drawby(25,1)
440   GOSUB 500
450 NEXT S
460 plot centre(150,54)
470 GOTO 340
480
490
500 REM Delay loop
510 FOR X = 1 TO 270
520 NEXT X
530 RETURN
```

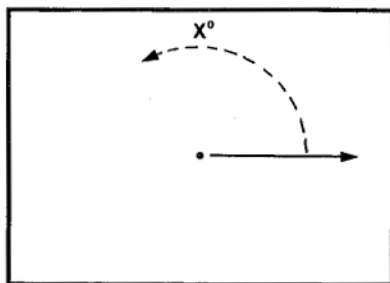
The program follows the pattern set in the small program illustrated earlier in this section. Note the delay inserted at line 440: this makes the clock beat seconds (fairly) accurately.

ARC, FILL, AXES

These three commands are described in the Panel, and are used in the "Graphics Demo" program. If you wish to become proficient with graphics, your best plan is to write a program of your own using the commands we have used in this section so far. When you have made it run to your satisfaction, use the Panel and the "Graphics Demo" to explore the more powerful commands.

PANEL 16—GRAPHICS [2]

- DRAW (X,Y,C)** — draw a line from the current pen position to (X,Y) in colour (C). (For colour numbers, see notes on COLOUR below.)
- DEGREES** — tells the computer to accept direction instructions in degrees (You may also use RADIANS).
- TURN(X)** — sets the direction of the pen in degrees or radians.
The pen starts like this
and degrees are measured anti-clockwise



- TURNBY (X)** — turns the pen through (X) degrees. Positive numbers turn it anti-clockwise, negative clockwise.
- DRAWBY (X)** — draws a line of length (X) in whatever direction the pen is pointing. After the line has been drawn, the pen returns to the *original* point.
- MOVEBY (X)** — moves the pen distance (X) in whatever direction it is pointing, drawing as it goes.
- COLOUR(X)** — colour(1) is foreground colour, i.e. you can draw in it; colour(2) is background colour, i.e. you can use it for rubbing out; colour (3) draws the opposite of whatever it is on top of.

PANEL 16—CONTINUED

- ARC(d,a) — moves the pen distance (d) while turning it through angle (a). Remember to set the direction of the pen before using this command.
- FILL — fills in the area around the pen, i.e. to fill in an area, place the pen within it and use plot fill. Warning: make sure there are no gaps in the lines surround the area you wish to fill.
- AXES(x,y) — draws and marks axes (i.e. vertical and horizontal lines through a point) crossing at the point where the pen is placed. x and y are the intervals at which the axes are to be marked. If you do not want intervals marked, set x and y to 0.

TO PLOT TEXT CHARACTERS ON A GRAPHICS SCREEN

- place the pen where you want the text to start;
- use PLOT instead of PRINT

SECTION 17—DATA

So far in this Guide, we have put data into programs by means of the INPUT statement. But there is another way.

Suppose we need a certain set of numbers every time we use a program or a part of a program and we do not want to INPUT them every time. The DATA statement looks after this.

Try the following.

```
10 DATA 55,45,40,25,55,25,55,10
20 DATA 40,10,40,25,19,39,34,60
30 DATA 55,45,75,45,75,25,55,25
40 OPEN#0,0,"110"
50 OPEN#1,11,"n190"
60 plot background(1),wipe
70 plot range(100,75)
80 plot place(55,25)
90 FOR I = 1 TO 12
100 READ X,Y
110 plot move(X,Y)
120 NEXT I
130 END
```

The result of this program may not be startlingly original, but it makes a point. The READ statement in line 100 is functioning in much the same way as an INPUT. It is taking the DATA numbers, two by two, and setting X and Y to these values, which are then used in the program.

NOTE: the READ statement not only uses the data numbers, it *uses them up*. If you want to use them again in the program you have to insert the command RESTORE. Starting a program with RUN automatically restores all data.

PANEL 17—DATA

To set up data, use the word DATA, followed by the items, divided by commas.

```
10 DATA 90,55,72,16,122,123,246,5
```

Items may be numeric or string

```
20 DATA NAME,ADDRESS,POSTCODE,AGE,SEX,  
MARRIED
```

or even both together

```
30 DATA SPEED,30,40,50,70,MPH,40,110,KM
```

To use data, use the command READ

```
50 READ X
```

```
50 READ X$
```

```
50 READ S$,A,B,C,D,C$
```

Items in READ statements must match those in the corresponding DATA statements. For example

```
10 DATA ONE,2,THREE,4
```

```
100 READ A$,B,C,D
```

will not work. Line 100 should be

```
100 READ A$,B,C$,D
```

To re-use data: use the command RESTORE

```
500 RESTORE — restores all data in the program
```

```
500 RESTORE 20 — restores data beginning with the first  
item on line 20, and continuing on  
subsequent lines.
```

SECTION 18—RND, INT, ETC.

We started this Guide with the idea that computers are incredibly precise, so that we have to “talk” to them in precise terms or we shall get the most ridiculous results.

The NIM program was an illustration of that precision: it is impossible for the computer to lose. Yet as a matter of fact it is quite possible to make computers act imprecisely, if we give them precise instructions to do so. This kind of behaviour is related to their ability to produce *random numbers*.

Type `print rnd` `NL`

and you get a decimal number between 0 and 1. Do it again and you get another. Most of the time, however, random numbers are most useful if they are larger than one. No problem: type

```
PRINT RND * 10 NL
```

If you want whole numbers rather than decimal, type

```
PRINT INT(RND * 10)NL
```

INT, of course, means “integer” — but remember its action is rather crude: it simply cuts the decimal part off and so rounds the number *down*.

But there is more to RND than meets the eye. Type in the following program and run it.

```
10 FOR I = 1 TO 10
20 x = INT(RND * 10)
30 PRINT x,
40 NEXT I
50 END
```

Run it several times, without clearing the screen, and compare the results. It may come as a surprise to find that each set of “random” numbers is the same! In fact, RND is not truly random, but a huge series of numbers which for most purposes will do equally well.

To avoid this problem, NewBrain offers another command: RANDOMIZE. This starts the series off at a different point each time. Try it by adding to the program

5 RANDOMIZE

and test the result.

If you want to check how good the randomization is, the following program will do it for you. It checks how often the numbers 1, 2 and 3 come up.

```
10 RANDOMIZE
20 A = 0: B = 0: C = 0
30 FOR X = 1 TO 10
40   FOR I = 1 TO 100
50     R = INT(RND * 4)
60     IF R = 0 THEN GOTO 50
70     IF R = 1 THEN A = A + 1
80     IF R = 2 THEN B = B + 1
90     IF R = 3 THEN C = C + 1
100   NEXT I
110   PRINT "Sample";X;";";100*X;
120   PRINT "numbers:";A; B; C
130   PRINT
140 NEXT X
150 END
```

NOTES

- a) RND is multiplied by 4, not 3, because INT rounds down.
- b) Similarly, we have to exclude 0, which would result from rounding down anything less than 1.


The program takes about 30 seconds to execute.

Having found a way to generate 1, 2, and 3 randomly, we are now in a position to do something about the unfairness of the NIM program, which the computer always wins, because the nature of the game is such that if you make the first move and play perfectly you must win.

So why not mar the perfection of the computer's reasoning by causing its moves to be based on chance? The program "NIM 2" on the tape works in this way — at least it gives mere humans a chance!

The randomizing sequence follows line 256, but be warned: the nearer the computer gets to the end of the game, the smarter it _____ becomes.

PANEL 18—USEFUL FUNCTIONS

RND	— gives a random number between 0 and 1
RANDOMIZE	— varies the start of the random number series
INT	— rounds a number down to the integer below it
SQR(X)	— means $\sqrt{\quad}$
	— for example, $X\uparrow 3$ means X^3
PI	— means π (i.e. 3.141592654)
ABS(X)	— strips the minus sign from a number (for example $ABS(-22) = 22$)
SGN(X)	— gives a value of -1 if X is less than 1; + 1 if X is more than 1; and 0 if X is 0.

SECTION 19—DATA FILES ON TAPE

You probably felt a little disappointed with the second "League Table" program. Interesting though it is to make a computer up-date results and re-arrange them, the trouble is that as soon as you turn the computer off, you lose all your data. Computers ought to be able to store data away permanently, so that you can retrieve it and up-date it, and store it away again, as often as you like.

In large computer systems, this is normally done by storing the data on magnetic disks, but there is no reason at all why we should not do it with cassette tape. Admittedly tape is slower to use than disks, but it is no less reliable and the procedure is not especially difficult.

The easiest way to understand what is involved is to experience a program which uses a *data file on tape* for yourself. Get a spare cassette with some blank tape on it. Then load "League Table 3" from the *Beginners Guide* tape.

Run it, following the instructions on the screen, and you will see how useful such a system can be. Ours uses only five items and saves and retrieves one file, but the same kind of program could be expanded to many items and several files.

So how does it work? The key command for creating the data file is in line 1030.

```
940 REM Write files onto tape
950 PUT 31: PRINT
960 PRINT"1. Make sure you have some"
970 PRINT"   blank tape in the recorder"
980 PRINT
990 PRINT"2. Set recorder to 'RECORD'"
1000 PRINT
1010 PRINT"3. Then press NEW LINE"
1020 INPUT S$
1030 OPEN OUT#99,1,"Table"
1040 FOR I = 1 TO 5
1050   PRINT#99,X$(I)
1060   PRINT#99,Y(I)
```

```

1070 NEXT I
1080 CLOSE#99
1090 PUT 31: PRINT
1100 PRINT"Your league table is"
1110 PRINT
1120 PRINT"recorded in data file 'Table'"
1130 PRINT: PRINT "on the tape."
1140 END

```

The #99 is an arbitrary number: it would work equally well with 15 or 22 for example. What the command does is to open a "stream" through which data flows to "device" number 1 — hence the next figure. Device number 1 is the cassette recorder plugged into the TAPE 1 socket. Device number 2 would be a recorder plugged into the TAPE 2 socket. The word "Table" is the name we are giving the file, and again this can be changed at the programmer's choice.

The command PRINT in lines 1050 and 1060 is much the same as the normal command PRINT, except that the "printing" is done onto the tape instead of the screen, and the FOR ... NEXT loop makes sure that all elements in the arrays are printed.

Having opened a "stream" we have to close it again in line 1080. The reason for this is that the data does not in fact go directly from the main processing part of the computer onto the tape. Instead it is gathered together in an area of memory known as a "buffer", where it is made up into batches before being sent on to the cassette recorder. When the command comes to close #99, anything left in the buffer is sent out to the tape, so never forget to close the stream, or you will lose data.

Getting the file back again from the tape involves much the same process.

```

780 REM Get files from tape
790 PUT 31: PRINT
800 PRINT "1. Wind the tape back to"
810 PRINT "   before the file: 'Table'"
820 PRINT
830 PRINT "2. Set the cassette recorder"
840 PRINT "   to 'PLAY': PRINT
850 PRINT "3. Press NEW LINE"

```



```

860 INPUT S$
870 OPEN IN#99,1,"Table"
880 FOR I = 1 TO 5
890   LINPUT#99,N$(I)
900   INPUT#99,P(I)
910 NEXT I
920 CLOSE#99: GOTO 370

```

Line 870 opens stream #99 and the file "Table". Note that whereas before we opened "out", now we open "in", because the data is flowing inwards from the tape to the computer. You will probably not be surprised to see the INPUT in line 900, but the LINPUT in line 890 is something else! In fact it is a special form of INPUT which avoids problems when inputting string items from tape. A normal INPUT would not accept strings with commas in them, for example. Again we close the file in line 920.

There is no exercise in this section. Having seen how useful data files on tape can be, sooner or later you will find an application for them. So here are a few hints.

The example shows data files with one-dimensional string and numerical arrays. Using them with two-dimensional arrays is just as straightforward. You have only to manage the arrays with FOR ... NEXT loops, as in the ENCODE and DECODE programs, and save the information into the data files you create.

If you do not need anything as complex as arrays, and merely want to save a collection of string variables and/or numeric variables, you can store them in the same way. For example:

```

100 PRINT#99,A$
110 PRINT#99,B$
120 PRINT#99,C$
130 PRINT#99,X
140 PRINT#99,Y
150 PRINT#99,Z

```

This will save the contents of three string and three numeric variables into any data file you care to create, and you can load them back again, with INPUT and LINPUT as we did in "League Table 3".

Two words of caution.

1. It is safest to PRINT and INPUT (or LINPUT) each item separately, as in the examples.
2. Always PRINT and INPUT (or LINPUT) your variables in the same order, or you will get some very confusing results in your program!

PANEL 19—DATA FILES ON TAPE

TO SAVE DATA IN A FILE

1. Load the data into variables or arrays.
2. Open a stream to Device 1, the cassette recorder (or Device 2, a cassette recorder plugged into TAPE 2). The stream number may be any you choose, unless you are already using it for something else.

open out #99,1,"Books"

will make a data file called "Books"

If you prefer you need not name your file at all, in which case ask the computer to load the first data file it finds on the tape when you come to retrieve your file. For example open out #99,1

3. Save the variables or arrays.

PRINT #99,A\$ PRINT #99,Z

PRINT #99,X\$(2) PRINT #99,P(3)

It is best to use a separate command PRINT for each variable or array.

4. Close the stream: CLOSE #99 (This is most important — see remarks in Section 19)

TO LOAD DATA FROM A DATA FILE

1. Wind the tape back so that the file can be read.
2. Open a stream to bring data from Device 1 (the cassette recorder).
open in #99,1,"Books"

PANEL 19—CONTINUED

3. INPUT the data into variables or arrays.

LINPUT #99,A\$ INPUT #99,Z

LINPUT #99,X\$(2) INPUT #99,P(3)

Use INPUT for numeric variables and arrays.

Use LINPUT for string variables and arrays.

ALWAYS INPUT IN THE SAME ORDER AS YOU SAVED

ALWAYS USE SEPARATE INPUTS FOR EACH VARIABLE OR
ARRAY

4. Close the stream: CLOSE #99

NOTE: if you prefer, you can ask the user of the program to name the file. The program can then store the name in a variable and use the variable to name the file. For example:

100 PRINT "Name the file"

110 INPUT X\$

120 OPEN OUT #99,1,X\$

and the file name will be whatever the user has INPUT.

You can also find out the name of a file that has been opened by asking for FILES \$. For example:

OPEN IN#1.; PRINT "FIRST FILE FOUND IS"; FILES \$;"!"

SECTION 20-EXTRAS AND EXPANSIONS

Your standard NewBrain will do a great deal, as you should have discovered by now. But its possibilities for expansion are even more impressive. The following is a selection.

1. THE PRINTER

There is a socket labelled PRINTER on the back of your NewBrain. You need a suitable lead (supplied by Grundy Business Systems).

The NewBrain regards a printer as a "device", like a cassette recorder, so you have to open a "stream" and name the device by number. The printer is device number 8, so it is easiest to remember if you open stream number 8 as well. The procedure for making your printer work is as follows.

1. Make sure the printer is switched off.
2. Type open #8,8 **NL**
3. Switch on the printer.

In theory you should set the "baud rate" too. That is the rate at which your printer will accept information. For example

open #8,8"4800" **NL** sets a baud rate of 4800.

In practice, if you use the printer recommended by Grundy Business Systems, this is not necessary, since if no rate is set the computer will assume it is "9600" which is right for that printer.

Once you have set it up, the printer is easy to use.

TO LIST PROGRAMS — LIST#8 **NL**

TO PRINT ANYTHING WITHIN A PROGRAM

100 PRINT#8,"ABCDEFGHJKLMNOPQRSTUVWXYZ"

The printer changes lines automatically after 80 characters. More detailed instructions on use and facilities will be supplied with your printer.

NOTE: you should not attempt to use the TAB command with the printer.

2. NEWBRAINS CAN TALK TO EACH OTHER!

If you have a friend who has a NewBrain too, you can pass programs between the two computers, provided you have the right lead. The lead plugs into the COMS sockets on both machines.

To send a program from one computer to the other

1. Type `open #9,9 [NL]` on BOTH machines
2. On the machine which has the program type
`save #9 [NL]`
3. On the other machine, type
`load #9 [NL]`

It is also possible to send data. For example, to send the string A\$, type `PRINT #9,A$ [NL]` on one machine and `LINPUT #9,A$ [NL]` on the other (it does not matter in which order). While the machine is waiting to send data, the screen will go blank. While sending, it will flicker.

3. EXPANSION MODULE

The Expansion Modules plug into the large socket at the back of your NewBrain. In addition to providing certain interfaces, they allow you

- a) to expand your computer's memory up to a maximum of 2048 k bytes.
(Expansion memory modules will take both RAM and ROM in increments of 64k bytes.)
- b) to link one NewBrain with up to 32 others, or even more via further NewBrains.
- c) to add a DISK MEMORY unit. Disks, like tape, are for long-term storage, but they are faster to operate, and more convenient, since they can locate files without having to check every file stored, as is necessary with tape.
Grundy Business Systems offers a unit which handles disks each capable of storing up to 1 megabyte of memory. A great deal of software is available on disks only.

- d) to use ROM software modules. These plug into Expansion Memory Modules and provide
 - other programming languages than BASIC;
 - “device drivers”, i.e. means of operating special equipment such as a graphics plotter;
 - special applications, such as text processing and statistics

4. BATTERY MODULE

The module contains a set of re-chargeable batteries capable of operating the NewBrain for at least an hour, so providing some protection against mains fluctuations, and offering the possibility of using the NewBrain on the train or anywhere else away from a mains supply. The batteries can be re-charged using the normal power unit. They can also be re-charged while the NewBrain is working.

CONCLUSION

If you have followed the *Guide* so far, you should now be in a position to write your own programs, with a reasonable awareness of what possibilities are open to you. But the *Guide* should still be useful to you for a while yet. Learning to program is a matter of understanding how to apply rules, but learning to program *well* means developing your ability to work in a new logical language, and that means practice. As you practise, use the Panels in the *Guide*, and the index, but above all your NewBrain microcomputer.

ERROR MESSAGES

Errors may be shown on the screen in three forms

ERROR 2 if you are entering commands directly from the keyboard without using line numbers;

ERROR 2 AT 100 which indicates an error on line 100;

ERROR 2 AT 100:3 which indicates an error on the third statement in line 100.

- 2 Arithmetic error. For example, your program may have tried to divide by zero.
- 3 You have forgotten to put END at the end of your program, so the computer has tried to find another instruction and failed.
- 4 Illegal line number. Line numbers above 65535 are not allowed.
- 6 Illegal value, for example in an array subscript. The value should be in the range 0 — 65535.
- 7 Illegal array subscript value. For example, you may have written
10 DIM A\$(22)
and subsequently
100 PRINT A\$(30)
- 10 The computer has run out of memory space.
- 14 You have used open and followed it by something other than #, or IN# or OUT#. (Look carefully — a common mistake is open "0,0")
- 17 You have typed in a numeric function wrongly. For example LOG (X-"A")

- 19 Wrong number of subscripts in an array element. For example, you may have dimensioned an array as a one-dimensional array and then used it as a two-dimensional array.
- 20 A wrong expression. For example $A\$ = 2$
(It should be $A\$ = "2"$)
- 21 Something unrecognisable in the expression
For example: $X = A?3$
- 26 The keyword you have used begins with something other than a letter — this is usually caused by trailing your finger over too many keys. For example *PRINT
- 28 You have used a statement involving ON (e.g. ON X GOTO 1000) but the line number is greater than the highest line number in your program.
- 29 Your program tells the computer to GOTO a line that does not exist.
- 30 Input error. For example, INPUT X is executed by the computer, and the computer waits for a *number* to be input. If the user inputs A, the computer replies with ERROR 30 and waits for the user to try again.
- 31 In a PRINT statement, the computer has reached a point where it expects a comma, or a semi-colon, but it has found something else. (Look carefully at your work here: the computer and you may not have the same expectations, but there is certainly something wrong!)
- 33 the computer has come to a RETURN without having met a GOSUB. Look carefully at your logic.
- 34 You have used a computed GOSUB or computed GOTO routine, but you have missed out (or partly missed out) the GOSUB or GOTO.
- 35 You have tried to type LIST followed by two numbers (for example LIST 10–100), but you have put something else in place of the hyphen. Alternatively, you have typed your listing command correctly, but have touched another key as you were pressing NEW LINE e.g. LISTa
- 36 Bad input: you have included a quotation mark or a comma in your input reply to the computer.
- 37 You have tried to TAB to column 0.
- 38 You have used a POKE command with a value greater than 255.

- 39 Your program has asked the computer to READ more data than you have in your DATA statement. (Have you tried to use data twice in the same program without restoring it?)
- 40 You have used a CALL command with an illegal item as a parameter.
- 41 The computer has come to a NEXT statement without having met a corresponding FOR statement.
- 42 Empty DATA line.
- 44 Illegal control variable in a FOR ... NEXT statement. For example
FOR A\$ = 1 TO 10
(Only numeric variables are allowed.)
- 45 There is an error *inside* the FOR ... NEXT loop that starts on the line number named.
- 46 You have missed the TO out of your FOR statement, or partly missed it out.
- 47 You have written a FOR ... TO ... STEP statement (or the computer thinks you have!) and STEP is missed out or partly missed out.
- 48 The computer cannot find the NEXT statement corresponding to the current FOR statement.
- 49 Illegal FOR ... NEXT loop nesting.
- 52 Something you typed has caused the computer to expect a comma, but there is no comma.
- 53 Something you typed has caused the computer to expect a colon, but there is no colon.
- 54 CLOSE not followed by #.
- 55 Keyword misspelt (for example PRIJNTO)
or
Equals sign not found where expected.
- 56 Open parenthesis (not found where expected.
- 57 Closing parenthesis) not found where expected.
- 63 You have used an IF ... THEN statement, but the item that follows the 'IF' part is neither a keyword nor THEN.
- 65 Closing quotation marks not found where expected.
(Check carefully. *You* may not expect them, but whatever you typed caused the computer to do so.)

- 68 The word `OPTION` is not followed by `BASE`.
- 69 You have used `OPTION BASE` after the array has been created.
- 70 You have followed `OPTION BASE` with something other than 0 or 1.
- 71 Your program is dimensioning an array that already exists. You may not re-dimension arrays.
- 72 Dimension too large. Arrays may have up to 5374 elements, if memory allows.
- 73 You have used dimension 0 after having specified `OPTION BASE 1`.
- 74 Error in formatter, other than error 75.
- 75 Number in formatter not in range 0 . . 255, or not present.
- 80 `DEF` not followed by `FN`.
- 81 Illegal user defined function name in `DEF` statement. E.g. `DEF FN$ = A$`.
- 83 No `DEF` statement for user defined function.
- 84 Redefinition of a user defined function to have a different number of arguments or references to an array with the wrong number of dimensions.
- 85 Expression too complex to evaluate, or user defined function references too deeply nested to evaluate.
- 87 You have stopped a program and attempted to continue, using `CONT`, but the program cannot continue.
- 88 `CLEAR` is followed by something illegal.
- 90 Device stream or port not in range 0 — 255.
- 91 You have asked the computer to `VERIFY` the recording of a program onto tape. The computer's reply is that the program has failed the verification test.
- 92 You cannot close stream 0.
- 93 The words `ON ERROR` are not followed by `GOTO` or `GOSUB`
- 94 You have used line number 0. This is not allowed.
- 95 You have used `VAL`, but the string involved is not a number.
- 96 You have tried to use `LINPUT` with a numeric variable.

- 97 After ON ERROR or ON BREAK you have specified a non-existent line number.
- 98 You have made an error in using a PUT command.
- 99 You have used up all the data in your program.
- 100 Insufficient memory to open stream.
- 105 You have attempted to use a stream that is not open. For example you have typed LIST #8 without having opened stream number 8.
- 106 No such device.
- 107 Device-port pair already open.
- 108 You are trying to open a stream that is already open. You may find it convenient to insert a command into your program which closes the stream before you open it. Closing a stream that is already closed is allowed.
- 109 System error. E.g. attempt to input from a printer.
- 110 Syntax error in parameter string.
- 111 Attempt to open device which requires mains power when no mains power connected.
- 112 The computer has run out of memory while trying to comply with a graphics command to FILL.
- 113 Linked stream not a screen device.
- 114 Requested height too large for memory available to the linked stream.
- 115 Linked stream has been closed.
- 116 Position off the screen illegal in this context.
- 117 Incorrect PLOT command.
- 118 Cannot use input from graphics device (use PEN instead).
- 119 Attempt to output to graphics device before input function completed.
- 120 Syntax error in baud rate parameter string.
- 121 Port number other than zero for serial device.
- 130 Tape read error hardware failure. Check tape recorder and leads.
- 131 Tape read error: attempt to read block into a buffer which is too small, or hardware failure.

- 132 Tape read error: hardware failure (Checksum error).
- 133 Attempt to read past the end of a tape file, or hardware failure.
- 134 Attempt to read a tape file out of sequence, or hardware failure.
- 135 Attempt to output a tape file opened for input or vice versa.
- 136 Syntax error in parameter string.
- 200 Time out error on software serial input port.

INDEX TO INFORMATION PANELS

	PANEL		
ABS	18	Cursor	2,4
AND	10	— moving down, up	8
ARC (Graphics)	16	— placing	6
Arrays		— use of in editing	9
— one-dimensional	13		
— two-dimensional	14	DATA	
AXES	16	— files on tape	19
		— statements	17
BACKGROUND (Graphics)	15	DEGREES (Graphics)	16
Battery module	20	DELETE (editing)	9
		Dimensioning arrays	13,14
Cassette recorder		Disk memory	20
— connecting up	1	DRAW, DRAWBY (Graphics)	16
— data files	19		
— loading programs	2	Editing	9
— saving programs	7	Elements in arrays	13,14
— verifying saving	7	Errors	
CENTRE (Graphics)	15	— numbers	APPENDIX
Characters		— on loading	2
— graphics characters	8	Escaping from a program	4
— on high-resolution		Expansion module	20
graphics	16		
— sets	8	Files, data, on tape	19
CHR\$	4	FILL (Graphics)	16
CLEAR	13	FOR ... NEXT	5
Clearing		Functions	18
— arrays	13		
— line	8	GOSUB ... RETURN	6
— part of screen	8	GOTO	3
— program from memory	3	— computed GOTO	6
— screen	3	Graphics	
CLOSE		— characters	8
— data file	19	— high resolution	15,16
— graphics screen	15	— text on	16
Colon	4	Graphics screen	15
COLOUR (Graphics)	16	Graphs, drawing	16
Comma	4		
Computed GOSUB, GOTO	6	Handling of strings	11,12
Concatenation of strings	10	HOME	
Connecting up	1	— to clear a line	3
Control characters	4	— to clear the screen	8

IF ... THEN	4	Power unit, connecting up	1
— use of AND / Or	10	PRINT	3
INPUT	4	Program	
Insert (Editing)	9	— catalogue on tape	7
INSTR	12	— changing a line	3
INT	18	— clear from memory	3
		— escaping from	4
Keyboard	Page 17	— LIST	3
		— RUN	3
Leads	1	— SAVE	7
LEFT \$	11	— STOP	3
LEN (length of string)	11	— transferring	20
Lines (program)		— VERIFY	7
— editing	9	Programming, starting	3
— replacing	3	PUT	
Lines (screen)		1 (insert blank line)	8
— clearing	8	10 (cursor down)	8
— 40 or 80 characters	8	11 (cursor up)	8
LINPUT	19	14 (for reversed-out characters)	8
LIST	3	22 (place cursor)	6
Loading		23 (character set control)	8
— data files from tape	19	30 (clear line)	8
— programs from tape	2	31 (clear page/screen)	4
Loops (FOR ... NEXT)	5		
		Random numbers	18
Mathematical functions	18	RANGE (Graphics)	15
Memory		READ	17
— additional RAM	20	Recorder, cassette	1
— disk	20	— data files	19
MID \$	11	— loading programs	2
Monitor, connecting up	1	— saving programs	7
MOVE (Graphics)	15	— verifying saving	7
MOVEBY (Graphics)	16	REM	5
		RESTORE	17
Nesting loops	5	RETURN (GOSUB ... RETURN)	6
NEW	3	Reversed-out characters	8
NEXT (FOR ... NEXT)	5	RIGHT \$	11
Numeric variables	3	RND	18
		ROM software modules	20
One-dimensional arrays	13	RUN	2
OPEN			
— for cassette recorder	19	SAVE	
— for graphics screen	15	— programs	7
— for long and short lines	8	— data files	19
— for page length	7	SCALE (Graphics)	15
OR	10	Screen	
		— clearing	3
PAGE	7	— graphics	15
PEN (Graphics)	15,16	— part clearing	8
PLOT (Graphics)	15,16	Semi-colon	4

SGN	18	THEN (IF ... THEN)	4
SQR	18	Transferring programs	20
Starting up	1	TURN, TURNBY (Graphics)	16
STOP	3	Two-dimensional arrays	13
Strings	10		
— concatenation	10	Variables	
— handling	11,12	— numeric	3
String variables	10	— string	10
		— value after loop	5
TAB	6	VERIFY	7
Tape, data files on	19		
Television set, connecting up	1	WIPE (Graphics)	15
Text on graphics screen	16		



Grundy Business Systems Ltd.

Sales and Administration

Somerset Road, Teddington,
Middlesex TW1 8TD
Tel: 01 977 1171

Marketing and R&D

Science Park, Milton Road,
Cambridge CB4 4BH
Tel: 0223 350355

The NewBrain Beginner's Guide was written by
Information Transfer Ltd, Cambridge